Incremental Learning of Linear Model Trees

Duncan Potts, Claude Sammut
duncanp@cse.unsw.edu.au, claude@cse.unsw.edu.au
School of Computer Science and Engineering
The University of New South Wales
Australia

UNSW-CSE-TR-0418
June 2004

THE UNIVERSITY OF
NEW SOUTH WALES

Abstract

A linear model tree is a decision tree with a linear functional model in each leaf. Previous model tree induction algorithms have operated on the entire training set, however there are many situations when an incremental learner is advantageous. In this report we demonstrate that model trees can be induced incrementally using an algorithm that scales linearly with the number of examples. Two incremental node splitting rules are presented, together with incremental methods for stopping the growth of the tree and pruning. Empirical testing in four domains ranging from a simple test function to a complex 13 dimensional flight simulator, shows that the new algorithm can learn a more accurate approximation from fewer examples than alternative incremental methods. In addition a batch implementation of the new algorithm compares favourably with existing batch techniques at constructing model trees in these domains. Moreover the induced models are smaller, and the learners require less prior knowledge.

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.

In this report we shall focus 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). The problem then becomes one of incrementally learning an approximation to the function f using the states experienced by the agent.

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 incremental learning techniques can be applied (Ljung, 1987). This transformation, however, requires detailed prior knowledge of the types of non-linearity present.

Standard parametric machine learning methods for incrementally inducing an unknown function f include neural networks, locally weighted learning (LWL) (Atkeson et al., 1997) and radial basis functions (RBFs). The training of neural networks can be slow, and is prone to local minima thus convergence is not guaranteed. It is also difficult to estimate in advance the required network structure to achieve a pre-specified degree of accuracy. LWL, or kernel regression, comprises a set of models that are spatially localised in the input space by a weighting function. The advantage is that the error function (e.g. the least squares criterion) can be minimised locally for each model, therefore avoiding a high-dimensionality minimisation problem. Alternatively RBFs can be used to expand the input space of the original learning data such that the resulting higher dimensional representation becomes linear

in the learning parameters. The error function is then minimised globally over this new representation. If the number of receptive fields are distributed uniformly over the input domain then both LWL and RBFs scale exponentially with the number of dimensions, and even covering a 4-dimensional space becomes difficult. In addition the range of the inputs and the 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 a LWL-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; Potts, 2004b).

The contribution of this report is to develop new incremental methods for growing linear model trees, including two alternative node splitting rules, a stopping rule and a method for pruning. The algorithm is empirically tested in four domains and compared with existing incremental, instance-based and batch methods. In this work control is not considered, and non-linear models are built solely as a system identification task.

Section 2 gives a brief overview of linear model trees and section 3 surveys related work on both the incremental induction of classification trees and the batch induction of linear model trees. Section 4 describes the algorithm and section 5 presents the experimental results. The results are discussed in section 6 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

$$y = f(\mathbf{x}) + \epsilon$$

where $f(\mathbf{x})$ is the corresponding component of $\dot{\mathbf{z}}$ and $\mathbf{x} = [\mathbf{z}^T \ \mathbf{u}^T \ 1]^T$ is a $d = n+m+1$ dimensional column vector of regressors (the constant regressor is added to simplify the notation). 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 $\hat{f}(\mathbf{x})$ to $f(\mathbf{x})$ that minimises some cost function (e.g. sum of squared errors) over a set of training examples.

For a linear model tree the estimate $\hat{f}(\mathbf{x})$ is a decision tree with a linear model in each leaf (see figure 1). The decision tree partitions the input space and within each leaf

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

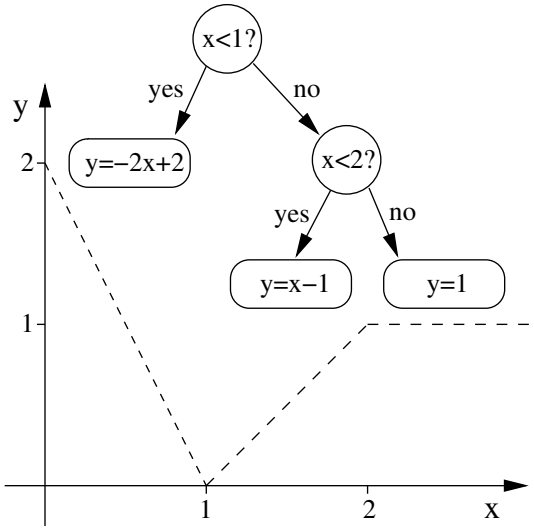


Figure 1: Example of a simple linear model tree with one regressor x.

where $\hat{\boldsymbol{\theta}}$ 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 $f(\mathbf{x})$ 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

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

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

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

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

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

If the true function $f(\mathbf{x})$ 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. The great advantage of using this technique is that the recursive least squares (RLS) algorithm can incrementally calculate $\hat{\boldsymbol{\theta}}_{\text{LS}}$ and RSS in time $O(d^2)$ for each new example. 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 incremental algorithm that is guaranteed to be fast. Insights into how an incremental method can be developed are then gained from the literature on constructing linear model trees in a batch setting.

3.1 Incremental Tree Induction

Utgoff et al. (1997) consider the incremental induction of classification trees where each leaf determines an element from a finite set of classifications. One of their aims is that the incremental 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 is 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. In this work we take a pragmatic approach and instead of restructuring, the tree is either allowed to carry on growing or is pruned back to where the incorrect split has been identified. Although this approach may result in larger trees or slower learning, it means we do not have to use a semi-incremental method that stores all examples and fast processing can therefore be guaranteed. The experiment in section 5.1.5 verifies that our new algorithm is not overly sensitive to different input distributions.

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. Leaves containing linear models are also not considered.

3.2 Batch Induction of Linear Model Trees

The general 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

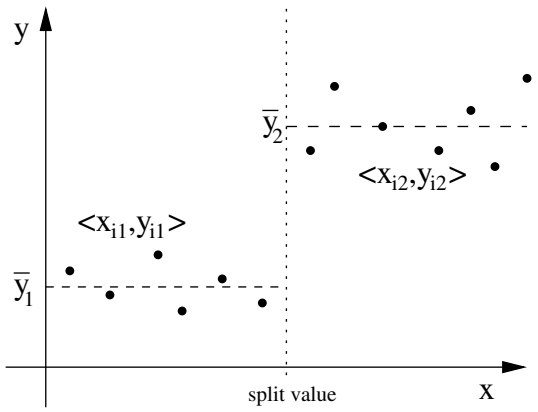


Figure 2: Splitting the data successfully into two subsets that are accurately characterised by their means.

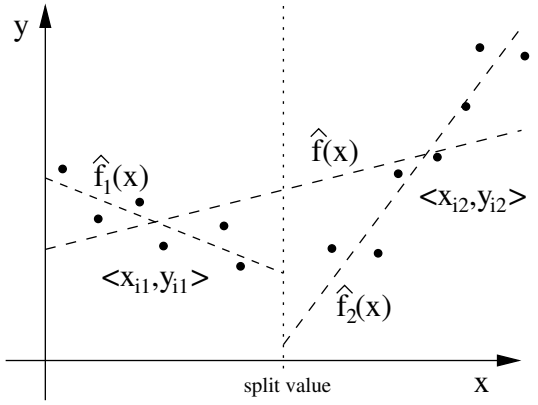


Figure 3: Splitting the data successfully into two subsets that are accurately characterised by fitted linear models.

for the induction of linear model trees. Denote the N examples at a particular node as $\langle \mathbf{x}_i, y_i \rangle$. A potential split divides these examples into two subsets. Denote the N_1 examples in the first subset $\langle \mathbf{x}_{i1}, y_{i1} \rangle$, and the N_2 in the second subset $\langle \mathbf{x}_{i2}, y_{i2} \rangle$.

Figure 2 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 $\bar{y}_1$ and $\bar{y}_2$. 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; Frank et al., 1998; Wang & Witten, 1997) chooses 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, and are therefore suitable when the leaf contains a constant value.

Figure 3 on the other hand shows a fitted linear model $\hat{f}_k$ constructed for each subset k on either side of the candidate split. If the leaf is to contain a linear model then clearly distance to the mean value is now an inappropriate measure of error, and distance to the linear regression plane should be used instead. This more suitable measure is used by RETIS (Karalic, 1992) which selects the split that minimises the total RSS over the two subsets

$$\mathrm{RSS}_1 + \mathrm{RSS}_2 = \sum_{i=1}^{N_1} \left(y_{i1} - \hat{f}_1(\mathbf{x}_{i1}) \right)^2 + \sum_{i=1}^{N_2} \left(y_{i2} - \hat{f}_2(\mathbf{x}_{i2}) \right)^2 \tag{5}$$

The minimisation of (5) 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 values increases with the number of examples (assuming the regressors are drawn from a continuous set), and it quickly becomes intractable to select from all possible splits with this method.

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. (2001) build up a multivariate linear model using a sequence of simple linear regressions hence simplifying the split selection, however splits near the root are therefore only selected on the basis of a few regression components and comprehensibility is lost due to the transformation of regressors. Dobra and Gehrke (2002) determine the split by separating two Gaussian clusters in the regressor attributes enabling oblique splits to be found. Li et al. (2000) also find oblique splits using principal Hessian directions, however the procedure is complex and requires interaction from the user and an iterative process to find the split value. Generally non-orthogonal splits make the splitting process even more complicated and they are therefore not considered in this work.

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 incremental algorithm, however, it is desirable to limit the growth of the tree in the first place and avoid any complex 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. There is clearly no need to make any split if the examples in a node can all be explained by a single linear model, however the node should be split if the examples suggest that two separate linear models would give significantly better predictions.

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 significant, then no splitting should occur until further evidence is accumulated.

4 Incremental Induction Algorithm

Section 4.1 presents two alternative splitting rules that can be used in the new incremental model tree induction (IMTI) algorithm. Sections 4.2 and 4.3 describe incremental methods for stopping the growth of the tree and pruning. Section 4.4 explains how smoothing is used to obtain more accurate gradient estimates, section 4.5 explains how categorical attributes can be incorporated into the tree and section 4.6 analyses the complexity of IMTI.

4.1 Splitting Rules

The first splitting rule is based on the difference in residual sums of squares (RD), and the second is an alternative approach that analyses the distributions of positive and negative

residuals (RA).

4.1.1 RD Splitting Rule

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 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 3; $\hat{f}(\mathbf{x})$ using all N examples, $\hat{f}_1(\mathbf{x})$ using the N_1 examples lying on one side of the split, and $\hat{f}_2(\mathbf{x})$ using the N_2 examples on the other side. The residual sums of squares are also calculated for each linear model, and denoted RSS, RSS_1 and RSS_2 respectively. The two smaller linear models will always fit the data at least as well, and $\mathrm{RSS}_1+\mathrm{RSS}_2 \leq \mathrm{RSS}$. However if the alternative hypothesis is true $\mathrm{RSS}_1+\mathrm{RSS}_2$ 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

$$F = \frac{(\mathrm{RSS} - \mathrm{RSS}_1 - \mathrm{RSS}_2) \times (N - 2d)}{(\mathrm{RSS}_1 + \mathrm{RSS}_2) \times d} \tag{6}$$

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 statistic with the smallest associated p-value (probability in the tail of the distribution). This method was used by Sicilano and Mola (1994) to grow linear model trees in a batch setting. Denote the smallest p-value as α. The key advantage of such a statistical test in an incremental implementation is that if the probability α is not small enough to discount H_0 with the desired degree of confidence, no split should be made at all.

The derivation of (6) requires that $\hat{f}(\mathbf{x})$ is an empty model upon node creation and that $\hat{f}_1(\mathbf{x})$ and $\hat{f}_2(\mathbf{x})$ are built from the same examples as $\hat{f}(\mathbf{x})$ (and therefore that $N = N_1 + N_2$). However in an incremental setting when a node is split and two children are created there is already a fitted linear model for each child (the models $\hat{f}_k(\mathbf{x})$ that were instrumental in choosing the split). Assume that for a particular node this initial linear model was created from N_0 examples ($N_0 = 0$ only at the root). As before a candidate split partitions additional examples into two sets: N_1 forming the model $\hat{f}_1(\mathbf{x})$ and residual sum of squares RSS_1 on one side, and N_2 forming $\hat{f}_2(\mathbf{x})$ and RSS_2 on the other. The model $\hat{f}(\mathbf{x})$ and residual sum of squares RSS is built from all $N = N_0+N_1+N_2$ examples. Under these more general conditions it can be shown (equation (A-1) in the Appendix with $p = 1$ and $q = 2$) that the statistic

$$F_{\mathrm{RD}} = \frac{(\mathrm{RSS} - \mathrm{RSS}_1 - \mathrm{RSS}_2) \times (N_1 + N_2 - 2d)}{(\mathrm{RSS}_1 + \mathrm{RSS}_2) \times (N_0 + d)} \tag{7}$$

is distributed according to the $\mathcal{F}$ distribution with N_0-d and N_1+N_2-2d degrees of freedom. This statistic is used in the incremental algorithm because it takes into account information already collected before the parent node was split.

The intractability of maintaining linear models on each side of every split value has already been noted, therefore a constant number κ of candidate split values is defined in advance for each of the $d-1$ regressors. In each leaf a linear model is maintained on both sides of the $\kappa(d-1)$ candidate split values, and α is calculated from these models. Table 1 defines the

steps needed to decide whether to split a leaf node given a training example. The stopping parameter δ is explained in section 4.2.

Table 1: RD leaf training function.

```
    function Train(leaf t, example ⟨x, y⟩)
 1   update leaf model m with ⟨x, y⟩ (using RLS)
 2   for each regressor i
 3       for each potential split j
 4           if x_i < split value s_ij
 5               update leaf model m_ij1 with ⟨x, y⟩
 6           else
 7               update leaf model m_ij2 with ⟨x, y⟩
 8           end if
 9           calculate F_RD using (7), and the associated p-value
10       end for
11   end for
12   calculate δ using (10) for the split with the minimum p-value (α)
13   if α < α_split and δ > δ_0
14       split the leaf
15   end if
    end Train
```

4.1.2 RA Splitting Rule

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 incremental RA splitting rule is based on the SUPPORT technique which is analysed below to show how the degree of confidence in the split is obtained and what approximations are required in an incremental implementation.

Assume that $\hat{f}(\mathbf{x})$ is the linear model constructed from all N examples at the node. The N_1 examples with non-negative residuals are put into subset 1, and the N_2 with negative residuals are put into subset 2. Label the regressor j for each example i in subset k as x_{ijk}. Let $\bar{x}_{jk}$ denote the mean of regressor j over the examples in subset k, and let s_j^2 denote the pooled variance estimate of regressor j over both subsets. The statistic

$$T_j^{(1)} = \frac{\bar{x}_{j1} - \bar{x}_{j2}}{s_j \sqrt{\frac{1}{N_1} + \frac{1}{N_2}}} \tag{8}$$

tests for the difference in means. Define $z_{ijk} = |x_{ijk} - \bar{x}_{jk}|$, let $\bar{z}_{jk}$ denote the mean of the z values for each subset, and let w_j^2 denote the pooled variance of the z values over both subsets. The statistic

$$T_j^{(2)} = \frac{\bar{z}_{j1} - \bar{z}_{j2}}{w_j \sqrt{\frac{1}{N_1} + \frac{1}{N_2}}} \tag{9}$$

tests for the 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 $\bar{x}_{j1}$ and $\bar{x}_{j2}$. Denote the probability under the Student's t distribution where $|t| > T$ as α. As for the RD splitting rule, a split is only made when α is small enough.

The advantage of this method is that only a single linear model is maintained within each leaf, thereby simplifying the training function (see table 2). The stopping parameter δ is explained in section 4.2.

Table 2: RA leaf training function.

```
    function Train(leaf t, example ⟨x, y⟩, δ)
1    update leaf model m with ⟨x, y⟩ (using RLS)
2    calculate T = max_{j,n} |T_j^(n)| using (8) and (9), and the p-value (α)
3    if α < α_split and δ > δ_0
4        split the leaf
5    end if
    end Train
```

Unfortunately neither t statistic can be calculated exactly in an incremental algorithm 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 requires an exact regressor mean $\bar{x}_{jk}$ which is also unavailable at intermediate stages. Our implementation uses the latest regression plane and regressor means to classify a residual as positive or negative and calculate z.

4.2 Stopping Rule

To illustrate the need for a stopping rule, assume a linear model tree is being grown to approximate a smooth convex function. As more examples are observed, the statistical splitting rule will repeatedly split the leaf nodes of the tree. The tree will grow without bound, fracturing the input space into a large number of linear models that together result in a very good approximation. Increasing the confidence limit on the statistical test will only slow this process.

It is, however, often desirable to limit the growth of the tree. The parameter

$$\delta = \frac{1}{s_y^2}\left(\frac{\mathrm{RSS}}{N-d} - \frac{\mathrm{RSS}_1 + \mathrm{RSS}_2}{N_1 + N_2 - 2d}\right) \tag{10}$$

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 N' training examples

$$s_y^2 = \frac{1}{N'-1}\sum_{i=1}^{N'}(y_i - \bar{y})^2 \qquad \text{and} \qquad \bar{y} = \frac{1}{N'}\sum_{i=1}^{N'} y_i$$

ensures that the stopping rule is invariant to scaling in the output values. 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 (see table 1). When using the RA splitting rule, δ cannot be calculated for each candidate split because RSS_1 and RSS_2 are not stored in the leaf and it would be too expensive to calculate them each time. Instead δ is calculated in the parent node, and passed down to the leaf as a parameter (see table 2).

4.3 Pruning

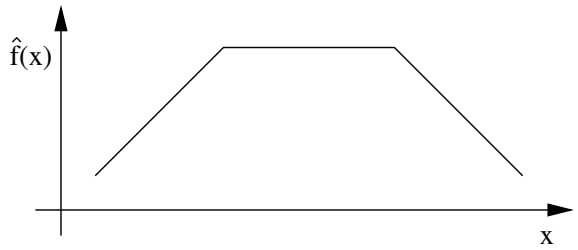
In the real world the statistical assumptions on which the splitting rules are based will inevitably be violated to a certain degree. It is therefore quite possible for bad splits to be made, and it becomes desirable to identify and prune branches of the tree that are not contributing to the overall accuracy. If the prediction accuracy of an internal node is deemed to be no worse than its corresponding sub-tree, then the sub-tree should be pruned. 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 of examples, 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 a fully incremental algorithm. However the third method is suitable because the sum of squared errors RSS is known in each leaf (and can easily be maintained in each internal node too).

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 for splitting allows for a more rigorous approach that can determine the probability that pruning will be beneficial. In section 4.1 the probability that all the examples in a node could have come from a single linear model is given by α. Therefore if α can be calculated at each internal node, it gives the probability that the node's entire sub-tree should be pruned. Pruning takes place if α at an internal node rises above a certain threshold α_{prune} (which must clearly be greater than α_{split}).

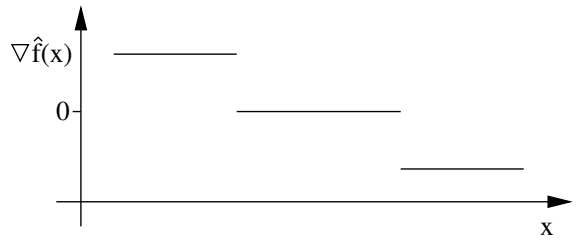
If a linear model is maintained at each internal node, α can be calculated from either the RD statistic (7) or the RA statistics (8) and (9). In fact a superior RD pruning statistic can be used that does not simply compare the internal node's model with the two models in the node's children, but with 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)

$$F_{\mathrm{RD}} = \frac{(\mathrm{RSS} - \mathrm{RSS}_{\mathrm{L}}) \times (N_{\mathrm{L}} - qd)}{\mathrm{RSS}_{\mathrm{L}} \times (N - N_{\mathrm{L}} + (q-1)d)} \tag{11}$$

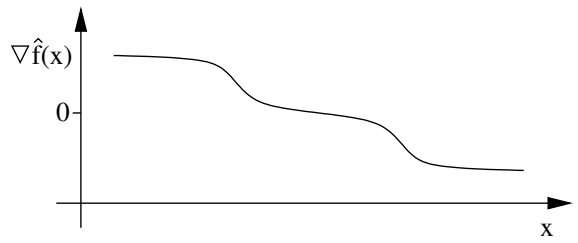
is distributed according to the $\mathcal{F}$ distribution with $N - N_{\mathrm{L}} + (q-1)d$ and $N_{\mathrm{L}} - qd$ degrees of freedom, where RSS is calculated from all N examples observed at the node, $\mathrm{RSS}_{\mathrm{L}}$ is the sum of the residual sums of squares in the leaves of the sub-tree, N_{L} is the total number of examples observed at the leaves and q is the number of leaves. In the experiments the statistic (11) is used to determine α at internal nodes when splitting with the RD rule, and the two RA statistics are used when splitting with the RA rule.



a. Linear model tree approximation to $f(\mathbf{x})$



b. Gradient of the approximation to $f(\mathbf{x})$



c. Smoothed gradient of the approximation to $f(\mathbf{x})$

Figure 4: Smoothing the discontinuities in the gradient with a Gaussian kernel.

4.4 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 $\mathbf{u}$. 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

$$\dot{\mathbf{z}} = f(\mathbf{z}_0, \mathbf{u}_0) + \frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}-\mathbf{z}_0) + \frac{\partial f}{\partial \mathbf{u}}(\mathbf{u}-\mathbf{u}_0) + \ldots \text{(higher order terms)} \tag{12}$$

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

$$\dot{\mathbf{z}} = \mathbf{F}\mathbf{z} + \mathbf{G}\mathbf{u} + \mathbf{c} \tag{13}$$

which have been intensively studied in control theory. From these equations we can determine the local stability of the system, and control policies to move the state along all feasible local trajectories.

Comparing (12) and (13) shows that $\mathbf{F}$ and $\mathbf{G}$ correspond to the derivatives of f. When f is approximated by a linear model tree (figure 4a) the gradient is constant within each leaf, and changes abruptly at the boundaries between leaves (figure 4b). These discontinuities result in poor gradient estimates of a function that is usually smooth over most of its input

domain. The estimates are therefore improved by smoothing with a Gaussian kernel which removes any discontinuities (figure 4c).

From the linear model (2) it is clear that the gradient estimate in each leaf k is simply the parameter vector $\hat{\boldsymbol{\theta}}_k$, hence the smoothed gradient estimate over all L leaves is

$$\frac{\partial \hat{f}(\mathbf{x})}{\partial \mathbf{x}} = \nabla \hat{f}(\mathbf{x}) = \frac{\sum_{k=1}^{L} w_k(\mathbf{x}) \hat{\boldsymbol{\theta}}_k}{\sum_{k=1}^{L} w_k(\mathbf{x})}$$

where $w_k(\mathbf{x})$ is the Gaussian kernel

$$w_k(\mathbf{x}) = \exp\left(-\frac{\rho}{2}(\mathbf{x} - \mathbf{c}_k)^T \mathbf{D}_k (\mathbf{x} - \mathbf{c}_k)\right)$$

The centre of the leaf in regressor space is $\mathbf{c}_k$, and the elements of the diagonal matrix $\mathbf{D}_k$ are the inverse squares of each leaf half-width component. The same smoothing process can also applied to the actual function estimates $\hat{f}(\mathbf{x})$.

4.5 Categorical Attributes

One advantage of the RD splitting rule is that categorical attributes can easily be incorporated into the decision nodes of the tree. 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 candidate split. The RD splitting rule can be applied without change to each categorical split.

The categorical attributes, however, cannot contribute to the linear regression in each node and the residual sums of squares are calculated using only the numerical attributes. It is also possible to constrain the numerical attributes so that they only contribute to either the splitting process or the linear regressions as in Loh (2002).

4.6 Training Complexity

As discussed at the start of section 2 a separate model tree is induced for each of the n components of $\dot{\mathbf{z}}$. Assuming that the stopping rule has limited the growth of each tree, the time taken to pass a training example down to a single leaf according to the tests in the intermediate nodes is bounded by the depth of the tree. For the RD splitting rule $O(\kappa d)$ models are updated in the leaf, and each RLS update takes $O(d^2)$. Hence the overall training complexity is $O(Nn\kappa d^3)$ 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 $O(Nnd^2)$. The algorithm therefore 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 (if the trees have stopped growing).

Although incremental algorithms can be sensitive to the order in which the training examples are presented, results in section 5.1.5 show that the tree size, and therefore the training complexity, does not vary a large amount with a shifting input distribution.

5 Experimental Results

The new incremental model tree induction (IMTI) algorithm with the RD and RA splitting rules is empirically tested in four domains:

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

Results are compared with the following algorithms (summarised in table 3):

1. A single linear model updated incrementally using recursive least squares (RLS).
2. The fully incremental adaptive locally weighted learning algorithm RFWR (Schaal & Atkeson, 1998). RFWR has since been adapted to improve its dimensionality reduction capability (Vijayakumar & Schaal, 2000), however the test domains do not contain redundant dimensions and the original algorithm, without any ridge regression parameters, is more competitive.
3. The batch CART algorithm using 10-fold cross validation and split selection that minimises the variance (Breiman et al., 1984), re-implemented by the authors.
4. The batch M5′ algorithm as detailed in Wang and Witten (1997) and implemented in WEKA[1].
5. The batch model tree induction algorithm SUPPORT with linear models in each leaf (Chaudhuri et al., 1994), re-implemented by the authors.
6. The batch equivalent of IMTI-RD that uses the RD splitting rule and the stopping rule of section 4.2 (Batch-RD).
7. The k-nearest neighbour instance-based approximator, with $k=10$ (10-NN). This algorithm stores all training examples, and returns the average of the 10 nearest neighbours when making predictions.

Both 10-NN and RFWR require a metric to be defined over the input space. For these algorithms each regressor is scaled to the range $(-1,+1)$ and the metric is the standard Euclidean distance. RFWR is also sensitive to an overall scaling of the function being approximated, and therefore each output component is also scaled to $(-1,+1)$ prior to training (with the exception of the 2D test function).

Training is performed using a single stream of non-repeating examples, such as would be obtained by an agent in the real world. The incremental algorithms (and 10-NN) can generate a single learning curve in one pass, whereas the batch algorithms must be re-run with a different number of examples to obtain each data point. Errors are measured using the normalised root mean square error, defined as

[1] See http://www.cs.waikato.ac.nz/∼ml/

Table 3: Algorithm differences.

Algorithm	Type	Representation
IMTI-RD / IMTI-RA	Incremental	Piecewise linear
RLS	Incremental	Linear
RFWR	Incremental	Piecewise linear
CART	Batch	Piecewise constant
M5′ / SUPPORT / Batch-RD	Batch	Piecewise linear
10-NN	Semi-incremental	Nearest neighbour

$$\text{nRMSE} = \sqrt{\frac{\sum_{i=1}^{N} \left(f(\mathbf{x}_i) - \hat{f}(\mathbf{x}_i) \right)^2}{\sum_{i=1}^{N} \left(f(\mathbf{x}_i) - \mu \right)^2}}$$

where $f(\mathbf{x})$ is the true function and μ is the mean of this function over the N examples in the test set. Results show the mean over 20 trials, and error bars and numerical errors reported in tables indicate one unbiased estimate of the standard deviation.

Parameters in RFWR and SUPPORT limit the number of linear models, resulting in a certain degree of asymptotic approximation error. These parameters are optimised in each domain to give the best performance while keeping the number of models to within a comparable range. Parameters are set as follows unless stated otherwise:

- The split error for the RD and RA rules $\alpha_{\text{split}} = 0.01\%$, thus a split is only made if there is less than a 0.01% chance that the data in the node came from a single linear model. This low level reduces the number of incorrect splits when testing many times. In addition a node is only split if each new leaf contains at least $3d$ examples.
- The IMTI pruning error $\alpha_{\text{prune}} = 0.1\%$.
- The number of candidates $\kappa = 5$ for the RD splitting rule.
- The RFWR initial distance metric $\mathbf{D}_0$ is optimised over the set $\{1\mathbf{I}, 2.5\mathbf{I}, 5\mathbf{I}, 10\mathbf{I}, 25\mathbf{I}\}$ ($\mathbf{I}$ is the identity matrix), the penalty γ is optimised over the set $\{10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}, 10^{-9}\}$ and the learning rates are optimised over the set {250, 500, 1000, 2500, 5000}.
- The SUPPORT parameters are MINDAT=30, v=10, f is optimised over the set {0.01, 0.02, 0.05, 0.1, 0.2} and η is optimised over the set {0.1, 0.2, 0.3, 0.4, 0.5}.
- IMTI, CART, SUPPORT and Batch-RD are all implemented with the smoothing technique described in section 4.4 and a smoothing parameter $\rho = 4$.

To avoid cluttering in the graphs, the batch algorithms and 10-NN are examined separately, and only the best method is compared with the incremental techniques.

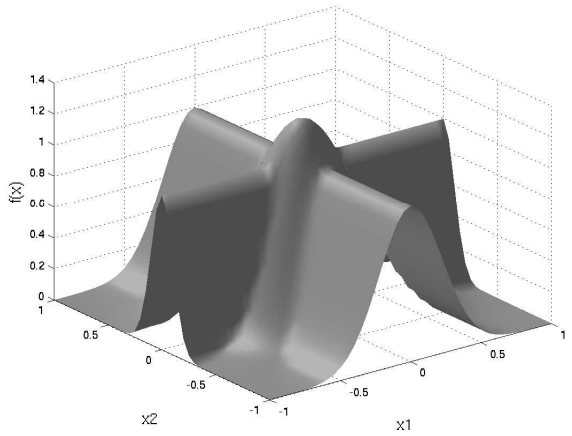
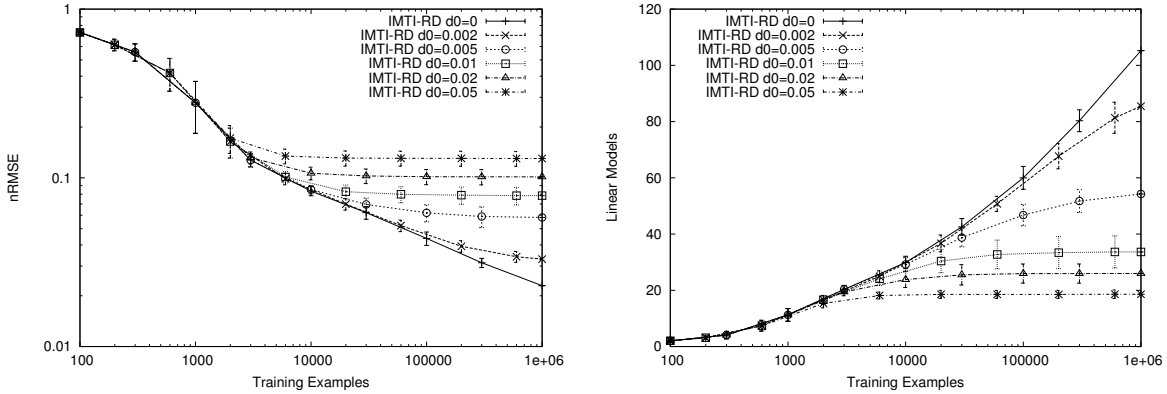


Figure 5: 2D test function.



a. Prediction error. b. Number of local models.

Figure 6: Effect of stopping parameter δ_0.

5.1 2D Test Function

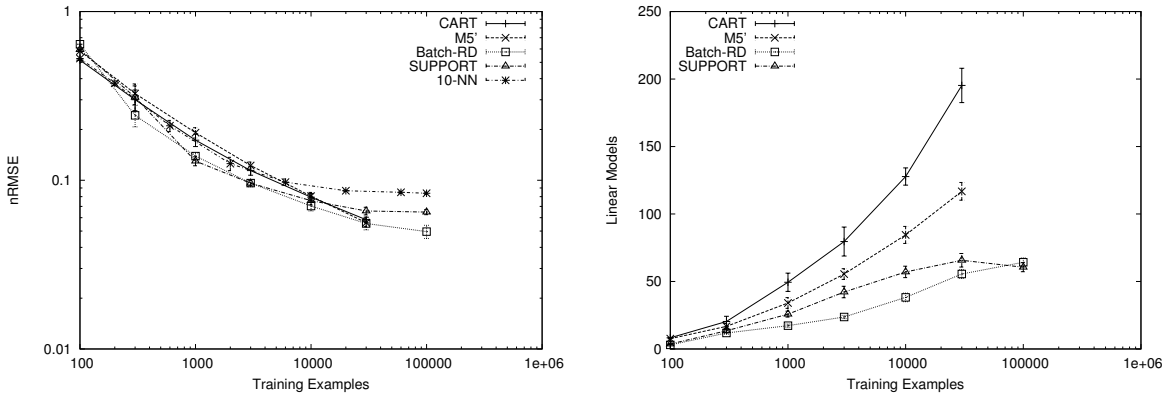
Initial tests are performed using the same function as Schaal and Atkeson (1998)

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

where ϵ is independent zero-mean Gaussian noise with variance σ^2 and $\sigma = 0.1$. This function can be pictured in figure 5. Training examples are drawn uniformly from the square $-1 \leq x_1 \leq +1, -1 \leq x_2 \leq +1$. The algorithms are tested using 2000 examples drawn in a similar manner, but without noise.

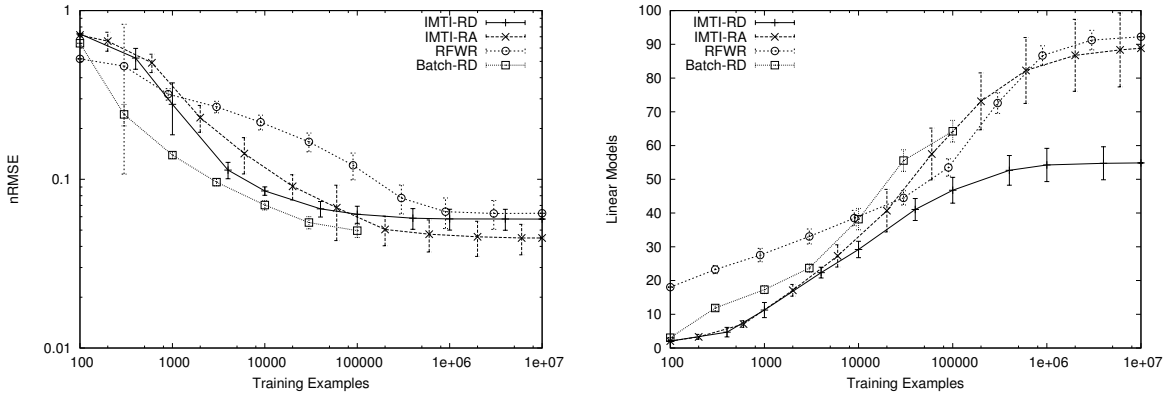
5.1.1 IMTI Stopping Rule

Figure 6 shows the effect of the stopping parameter δ_0 on the prediction error and size of model tree when splitting incrementally with the RD rule. It can be seen that the stopping parameter controls a trade-off between the asymptotic prediction error and the size of the model tree. A smaller value results in a larger tree and more accurate predictions. If the



a. Prediction error. b. Number of local models.

Figure 7: Comparison of batch algorithms and 10-NN on the 2D test function.



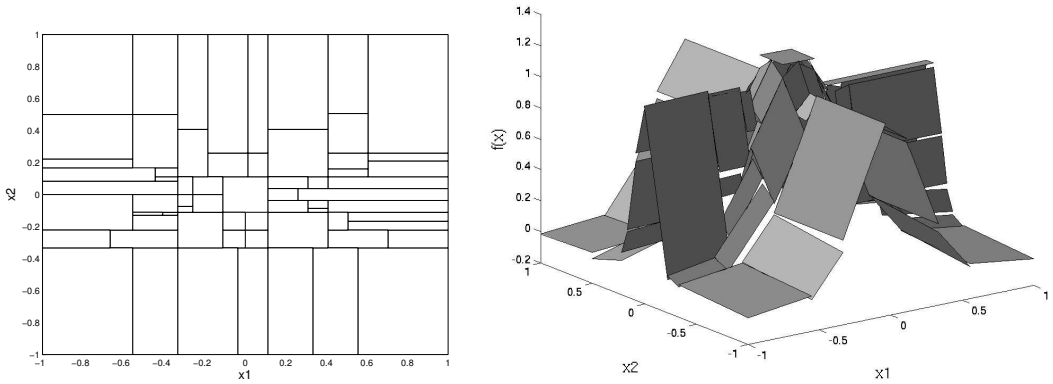
a. Prediction error. b. Number of local models.

Figure 8: Comparison of incremental algorithms on the 2D test function.

stopping parameter is zero the size of the tree grows without bound, and the predictions become more and more accurate (as discussed in section 4.2).

5.1.2 Batch Algorithms

CART uses the 1-SE pruning rule (which in this domain gives the same prediction accuracy as the 0-SE rule but with a smaller tree). M5′ has smoothing disabled, giving better results on this problem. The SUPPORT parameters are f=0.1 and η=0.4. Batch-RD uses a stopping parameter $\delta_0 = 0.003$. Figure 7 shows that Batch-RD learns a more accurate model tree using the same number of linear models as SUPPORT. CART and M5′ do not have parameters to limit the growth of the tree and it is not feasible to test them with more than 30,000 examples on our platform. Likewise it is not feasible to test Batch-RD, SUPPORT and 10-NN with more than 100,000 examples. In the remaining domains we do not consider CART and M5′.



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

Figure 9: Model tree induced by IMTI-RD.

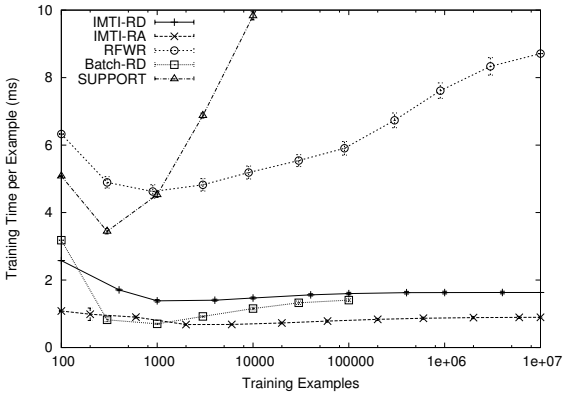
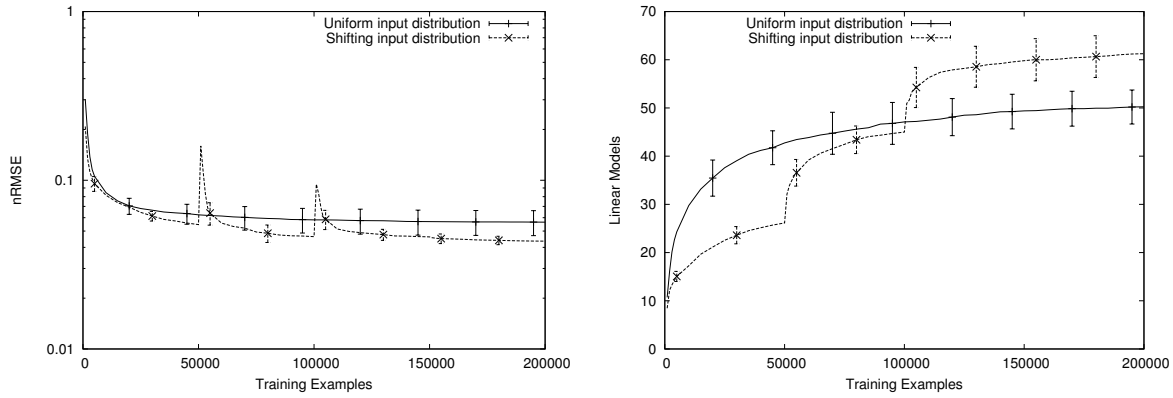


Figure 10: Performance on the 2D test function.

5.1.3 Incremental Algorithms

The stopping parameter $\delta_0 = 0.005$ for the RD splitting rule and $\delta_0 = 0.01$ for the RA rule (the difference arises because δ cannot be calculated in the leaf for the RA rule - see section 4.2). RFWR uses the parameters published in Schaal and Atkeson (1998) and learning rates set to 250. A single linear model can only achieve nRMSE = 1.0 on this function. Figure 8 shows that all the model tree algorithms learn a more accurate approximation and require fewer training examples than RFWR. The RA rule uses a similar number of local models, however the RD rule builds significantly smaller trees containing less local models. As expected the batch algorithm is superior to the incremental methods.

Figure 9a shows the partitioning of the input space by a typical model tree induced using IMTI-RD on this problem. The unsmoothed approximation formed by this tree is illustrated in figure 9b.



a. Prediction error. b. Number of local models.

Figure 11: Performance of IMTI-RD on a shifting input distribution.

5.1.4 Performance

Figure 10 compares the approximate training times per example for each algorithm (which also depend on the exact implementation details). The graph demonstrates the linear scaling of the IMTI algorithms and the computational advantage of IMTI-RA. Examination of the two batch algorithms shows that the Batch-RD stopping rule is more efficient and scales better with the number of examples than the expensive cross validation used in SUPPORT.

5.1.5 Shifting Input Distributions

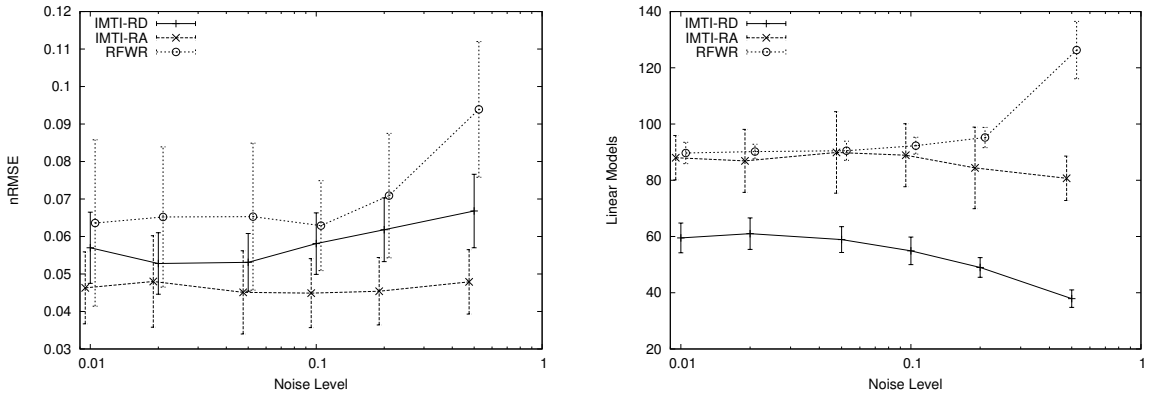
To test the resilience of the algorithm to a shifting input distribution we repeat the experiment in Schaal and Atkeson (1998) where the training data is presented in three distinct slightly overlapping batches. For the first 50,000 examples the training data and test data are uniformly drawn from $-1 \leq x_1 \leq -0.2$ (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 performance 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 11 compares the behaviour of IMTI-RD with a uniform input distribution as before and a shifting input distribution as described above. The two spikes in figure 11a show the temporary increase in error as the test and training distributions are shifted. However the error quickly decreases back to its previous level demonstrating that the algorithm is able to learn an accurate approximation to the new distribution while retaining the model associated with the old distribution. Figure 11b shows that the shifting distribution results in a small increase in model complexity.

5.1.6 Noise Resilience

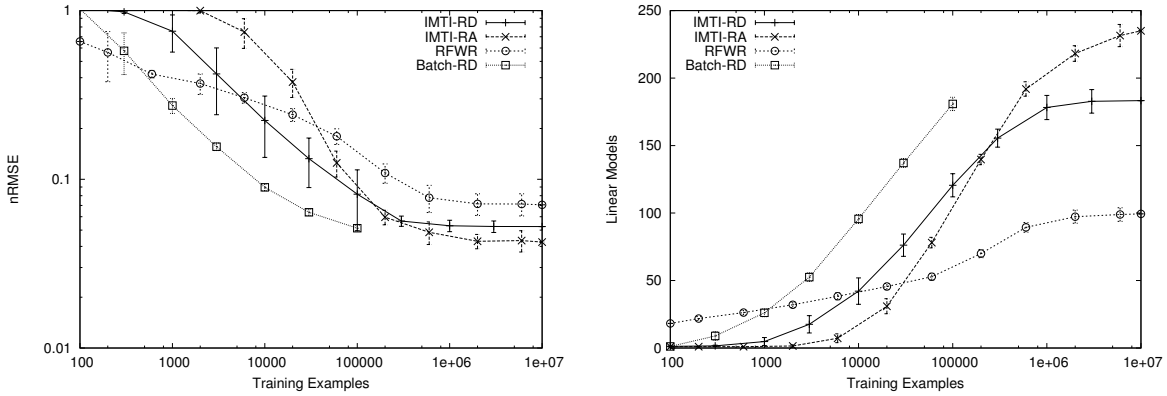
In this section we compare the behaviour of the incremental algorithms on the 2D test function with difference levels of noise varying from $\sigma = 0.01$ to $\sigma = 0.5$. Parameters are kept as before, and not individually tuned to each noise level.

Figure 12 compares the final prediction accuracy and number of local models for each algorithm after training with 10^7 examples. It can be seen that all three algorithms are fairly



a. Prediction error. b. Number of local models.

Figure 12: Sensitivity to noise on the 2D test function.



a. Prediction error. b. Number of local models.

Figure 13: Comparison of incremental algorithms on the rotated 2D test function.

insensitive to the noise level, although at the highest level IMTI-RD builds a smaller less accurate model and RFWR builds a larger less accurate model. The behaviour of IMTI-RA is clearly most desirable, followed by IMTI-RD and finally RFWR.

5.1.7 Rotated 2D Test Function

It is clear from figure 9b that the manner in which the test function is aligned to the axes favours the model tree algorithms. In this section we therefore present results from a similar 2D function rotated through 45°.

Figure 13 details the behaviour of the algorithms on this rotated function with the same parameters as before. Comparison with figure 8 shows that the approximations are equally as accurate, however the required number of linear models has increased substantially for the model tree algorithms, whereas it has remained almost constant for RFWR. Despite requiring a significantly more complex model when the target function is not aligned to the axes, the model tree algorithms still find efficient representations in the remaining experiments.

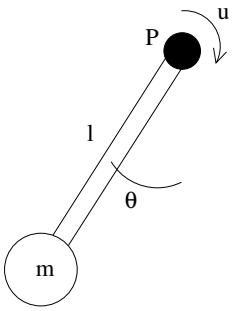


Figure 14: Pendulum.

5.2 Pendulum in Continuous Time

A pendulum rotating 360° around a pivot P (see figure 14) is a simple non-linear dynamic environment with a closed form for the gradient, allowing gradient errors to be examined. The dynamic model of the pendulum can be written

$$\dot{\mathbf{z}} = \begin{bmatrix} 0 & 1 \\ -\frac{g \sin\theta}{l\theta} & -\frac{\mu}{ml^2} \end{bmatrix} \mathbf{z} + \begin{bmatrix} 0 \\ \frac{1}{ml^2} \end{bmatrix} u \tag{14}$$

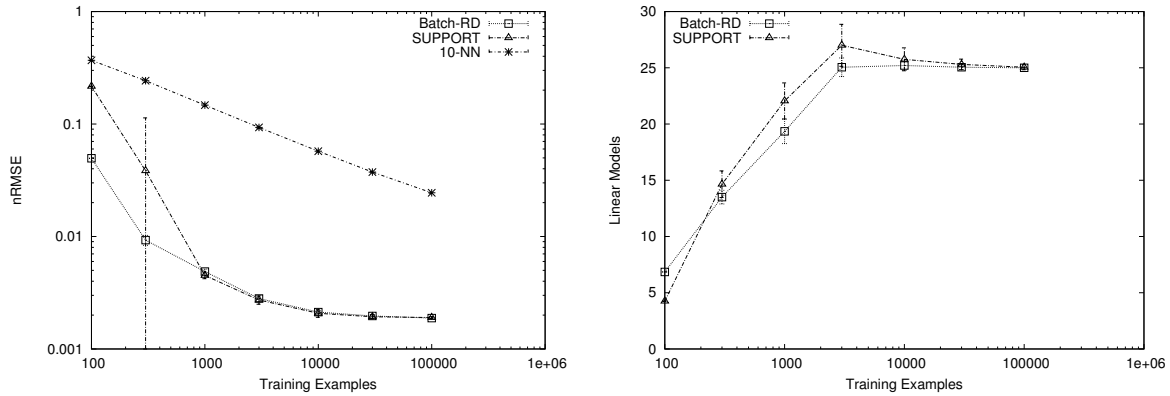
where $\mathbf{z} = [\theta \ \dot{\theta}]^T$, θ is the pendulum angle, g is gravity, $m = l = 1$ are the mass and length of the pendulum, $\mu = 0.1$ is a drag coefficient and u is the torque applied to the pendulum. Define $\mathbf{x} = [\mathbf{z}^T \ u]^T$ and $\mathbf{y} = \dot{\mathbf{z}} + \boldsymbol{\epsilon}$ 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.

In order to examine gradient predictions, we define the normalised root mean square *gradient* error over the N examples in the test set as

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

so that when the gradient estimate is correct everywhere nRMSE(Grad) = 0, and when the gradient estimate is zero everywhere nRMSE(Grad) = 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.

When the function being learnt has more than one component the prediction and gradient error reported is the mean over all components. In these cases a separate model tree is induced for each component and therefore each leaf model only predicts a single value. In contrast each local RFWR model predicts all components simultaneously, and therefore has correspondingly more parameters (see also the discussion in section 6). The number of local models reported for the model tree algorithms is the sum over all components.



a. Prediction error. b. Number of local models.

Figure 15: Comparison of batch algorithms and 10-NN on the continuous pendulum.

5.2.1 Batch Algorithms

The Batch-RD stopping parameter $\delta_0 = 2 \times 10^{-5}$ and SUPPORT uses the parameters f=0.1 and η=0.5. Figure 15 shows that Batch-RD learns more effectively from small sample sizes but both algorithms perform similarly for larger samples. The nearest neighbour algorithm is not competitive in this domain.

5.2.2 Incremental Algorithms

The stopping parameter $\delta_0 = 2 \times 10^{-5}$ for the RD splitting rule and $\delta_0 = 2 \times 10^{-4}$ for the RA rule. The RFWR initial distance metric $\mathbf{D}_0 = 5\mathbf{I}$, the penalty $\gamma = 10^{-9}$ and the learning rates are 100. A single linear model can only achieve nRMSE = 0.41 and nRMSE(Grad) = 0.76. Figure 16 shows that both IMTI algorithms learn the function and its gradient from fewer examples than RFWR, although the RA rule is slow to learn initially. Surprisingly IMTI-RD is almost as competitive as its batch equivalent Batch-RD. The local models built by both IMTI algorithms and Batch-RD estimate a single $\dot{\mathbf{z}}$ component and contain only 4 parameters, whereas the RFWR models predict both components and contain 8 parameters each. Figure 16c therefore shows that the overall number of parameters used by the IMTI algorithms is less than RFWR. These results demonstrate that IMTI-RD and IMTI-RA can effectively estimate the local linear approximations given by $\mathbf{F}$, $\mathbf{G}$ and $\mathbf{c}$ in (13) for the pendulum (14).

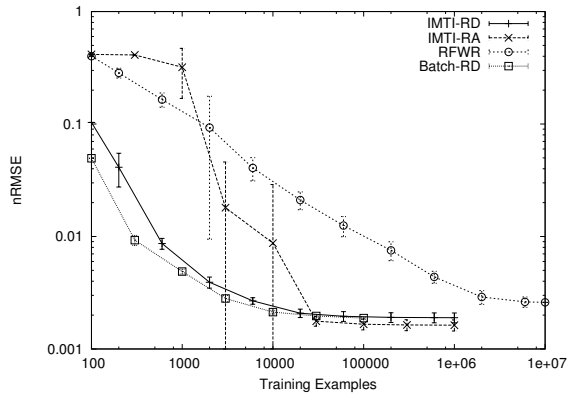
5.2.3 Performance

Figure 17 compares the performances of each algorithm showing again the computational advantages of IMTI-RA. As expected IMTI-RD, while still scaling linearly with the number of examples, is significantly slower in this higher dimensional problem.

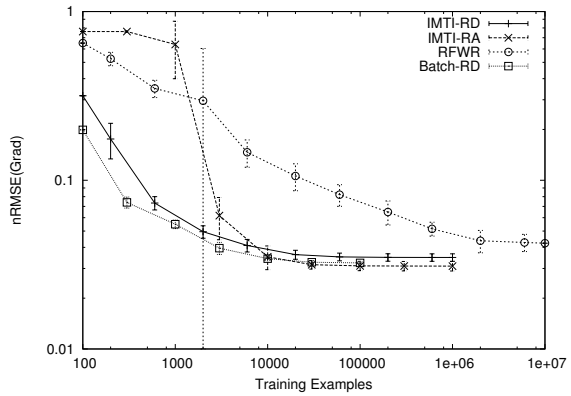
5.3 Pendulum in Discrete Time

The pendulum model can also be formulated in discrete time as

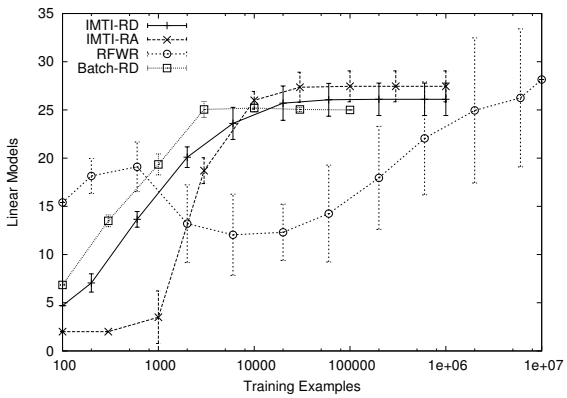
$$\mathbf{z}_{k+1} = \mathbf{\Phi}\mathbf{z}_k + \mathbf{\Gamma}u_k \tag{15}$$



a. Prediction error.



b. Gradient error.



. Number of local models.

Figure 16: Comparison of incremental algorithms on the continuous pendulum.

where k is the time step index. The observed state vector $\mathbf{y}_k = \mathbf{z}_k + \boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon}$ has the same characteristics as in section 5.2, and the regressors for $\mathbf{y}_k$ are the previous state $\mathbf{z}_{k-1}$ and action u_{k-1}. Examples of $\mathbf{x}_k = [\mathbf{z}_{k-1}^T \; u_{k-1}]^T$ are drawn uniformly from the same input domain as for the continuous time case. It is not possible to determine a closed form for $\boldsymbol{\Phi}$ or $\boldsymbol{\Gamma}$ 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. This also means that gradients cannot be calculated exactly, hence only prediction errors are examined in this section. The system identification task is to learn the model parameters $\boldsymbol{\Phi}$ and $\boldsymbol{\Gamma}$ given examples of $\langle \mathbf{x}_k, \mathbf{y}_k \rangle$. The algorithms are tested using 5000 examples drawn in a similar manner to the training data, but without noise.

5.3.1 Batch Algorithms

The Batch-RD stopping parameter $\delta_0 = 5 \times 10^{-6}$ and SUPPORT uses the parameters f=0.01 and η=0.4. Figure 18 shows that both algorithms make similar predictions although SUPPORT uses a highly variable number of local models. The nearest neighbour algorithm is not competitive in this domain.

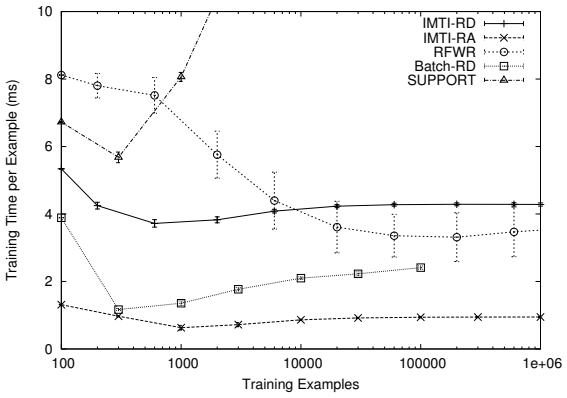
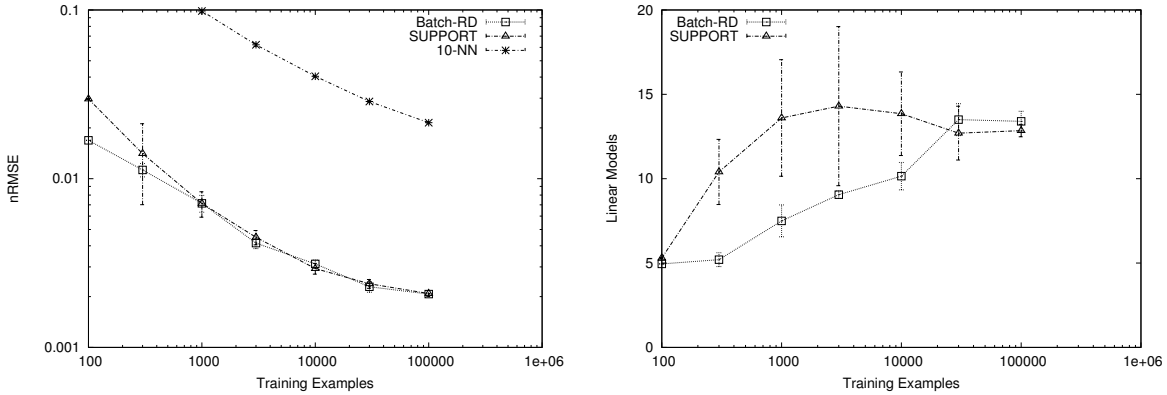


Figure 17: Performance on the continuous pendulum.



a. Prediction error. b. Number of local models.

Figure 18: Comparison of batch algorithms and 10-NN on the discrete pendulum.

5.3.2 Incremental Algorithms

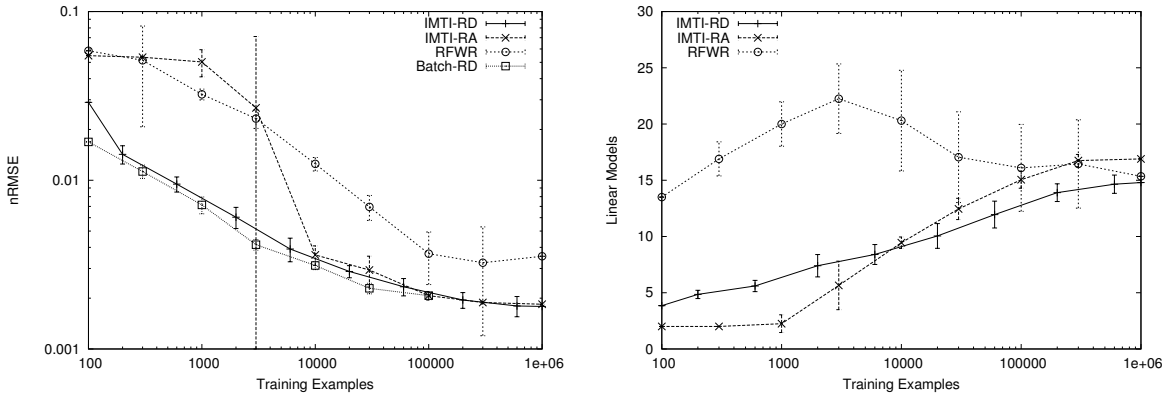
The stopping parameter $\delta_0 = 5 \times 10^{-6}$ for the RD splitting rule and $\delta_0 = 5 \times 10^{-5}$ for the RA rule. The RFWR initial distance metric $\mathbf{D}_0 = 5\mathbf{I}$, the penalty $\gamma = 10^{-9}$ and the learning rates are 5000. A single linear model can achieve nRMSE = 0.053. Figure 19 shows that as for the continuous time case, both IMTI algorithms learn faster than RFWR and use significantly less parameters overall (again each local RFWR model contains twice as many parameters), although again IMTI-RA suffers from an initial period of slow learning. Impressively IMTI-RD is as effective as its batch equivalent Batch-RD.

5.3.3 Performance

Figure 20 compares the performances of each algorithm and the results are similar to the continuous time case.

5.4 Cart and Pendulum

The problem of swinging up a pendulum on a cart demonstrates the behaviour of the algorithms in a more complex domain (see figure 21). The system is highly non-linear when the



a. Prediction error. b. Number of local models.

Figure 19: Comparison of incremental algorithms on the discrete pendulum.

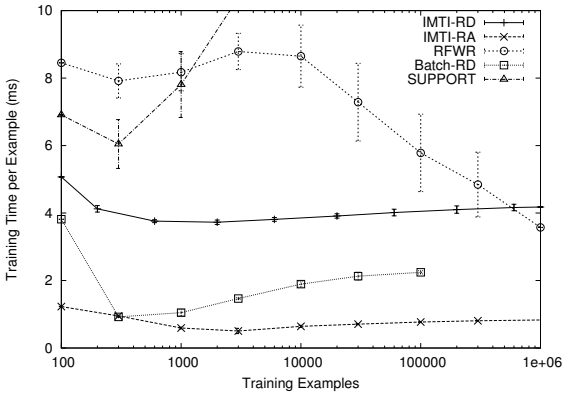


Figure 20: Performance on the discrete pendulum.

pendulum is allowed to rotate through 360°. The simplified dynamic model (not taking into account frictional effects) is

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

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 (15) 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 $\mathbf{y}_k = \mathbf{z}_k + \boldsymbol{\epsilon}$, 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 $\mathbf{z}_{k-1}$ and action u_{k-1}. The sequence of states generated is given directly to the learner without changing the order, so

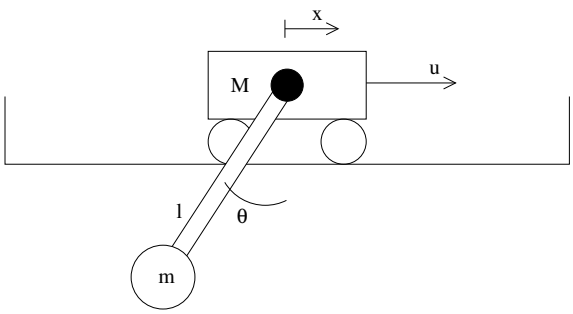
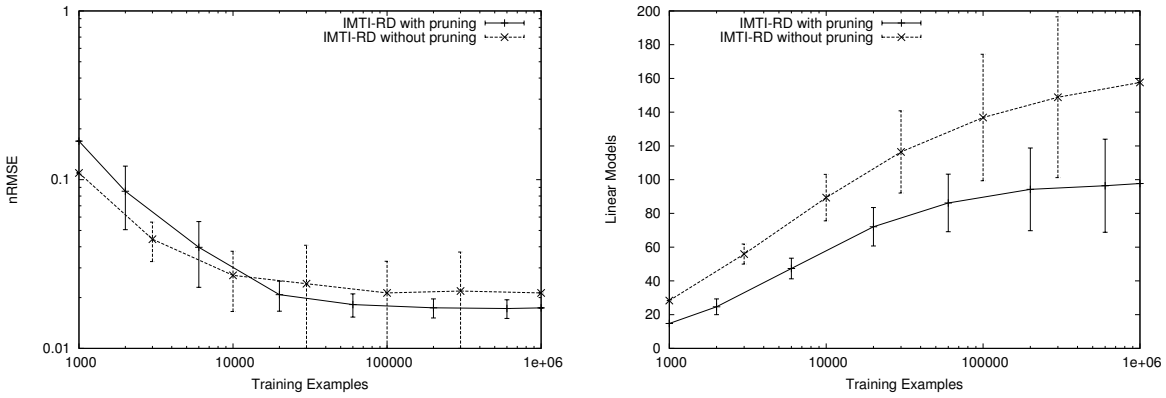


Figure 21: Cart and pendulum.



a. Prediction error. b. Number of local models.

Figure 22: Effect of IMTI pruning on the cart and pendulum.

that consecutive regressors are very highly correlated. The algorithms are tested using 10,000 examples randomly drawn from a similar sequence, but without noise.

5.4.1 IMTI Pruning

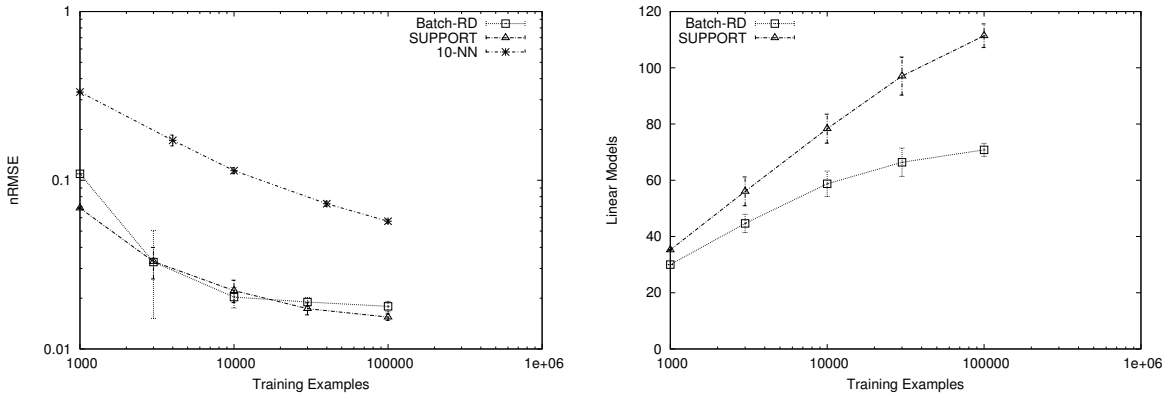
Figure 22 shows the effect of pruning for IMTI-RD with a stopping parameter $\delta_0 = 0.001$. It is seen to significantly reduce both the average size and variation in size across trials of the induced model trees and improve the prediction accuracy.

5.4.2 Batch Algorithms

The Batch-RD stopping parameter $\delta_0 = 5 \times 10^{-4}$ and SUPPORT uses the parameters f=0.2 and η=0.5. Figure 23 shows that both algorithms learn similar approximations although Batch-RD uses significantly less local models. The nearest neighbour algorithm is not competitive in this domain.

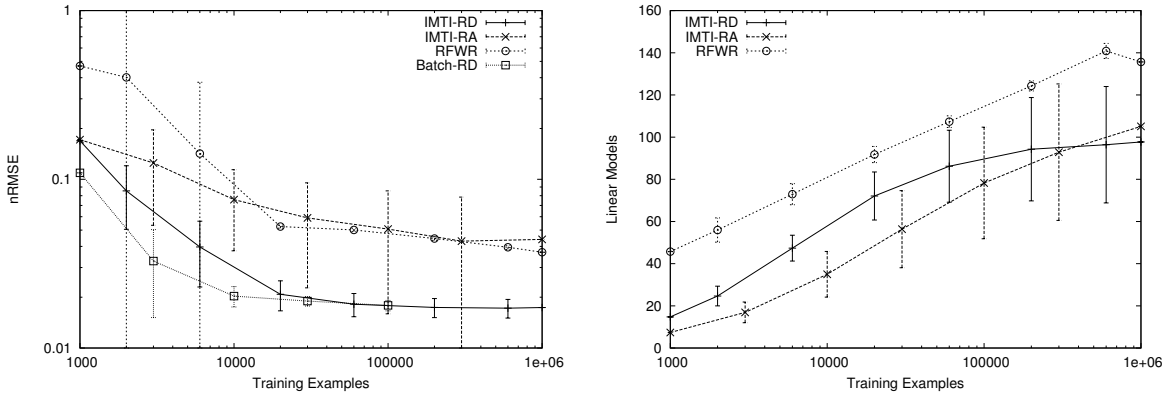
5.4.3 Incremental Algorithms

The stopping parameter $\delta_0 = 0.001$ for both the RD and RA splitting rules. The RFWR initial distance metric $\mathbf{D}_0 = 10\mathbf{I}$, the penalty $\gamma = 10^{-7}$ and the learning rates are 1000. A single linear model can only achieve nRMSE = 0.13. Figure 24 shows that IMTI-RD quickly



a. Prediction error. b. Number of local models.

Figure 23: Comparison of batch algorithms and 10-NN on the cart and pendulum.



a. Prediction error. b. Number of local models.

Figure 24: Comparison of incremental algorithms on the cart and pendulum.

converges to a more accurate approximation than RFWR, but the RA rule is unable to learn a consistent model over the different trials. The linear models built by the IMTI algorithms and Batch-RD estimate a single $\dot{\mathbf{z}}$ component and contain only 6 parameters, whereas the RFWR models predict all four components and contain 24 parameters each. Therefore IMTI-RD also uses significantly less overall parameters than RFWR.

5.4.4 Performance

Figure 25 shows that RFWR slows down a lot as the number of local models increases, but IMTI-RD hardly slows at all as its model tree grows.

5.5 Flight Simulator

Learning to fly an aeroplane is a complex high-dimensional task. These experiments use a flight simulator (see figure 26) 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 they progress to jet fighters. The

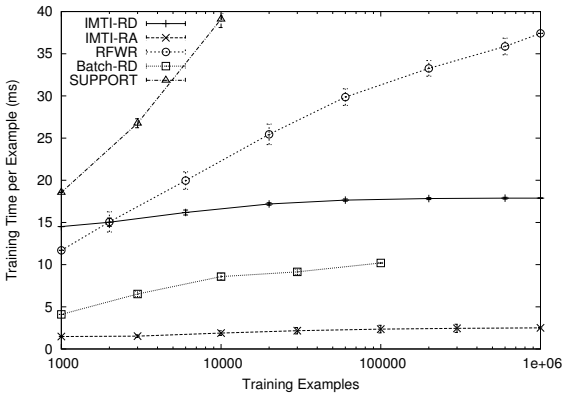


Figure 25: Performance on the cart and pendulum.

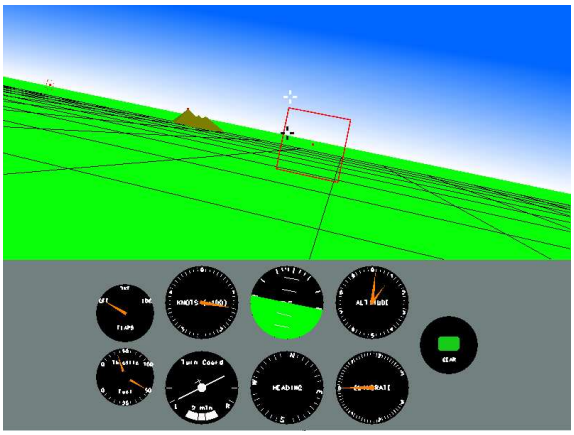
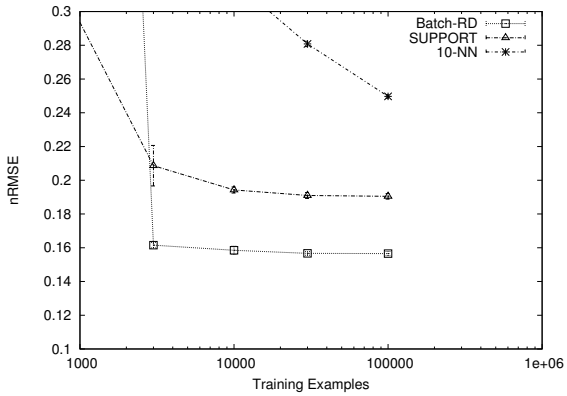


Figure 26: Flight simulator.

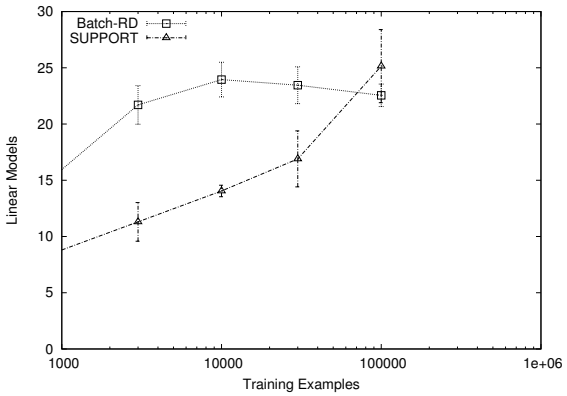
model was provided by the Australian Defence Science and Technology Organisation and is based on wind tunnel and in-flight performance data. The same simulator has also been used in previous work (Isaac & Sammut, 2003).

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.

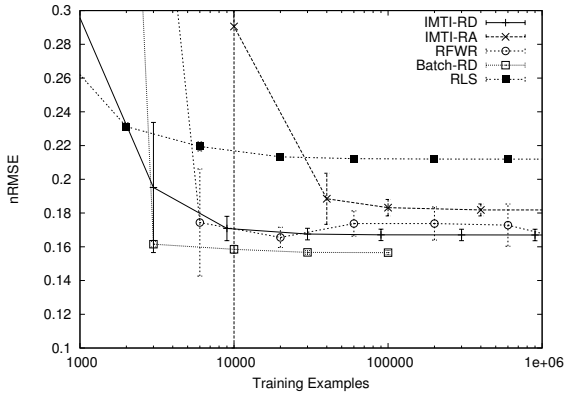


a. Prediction error.

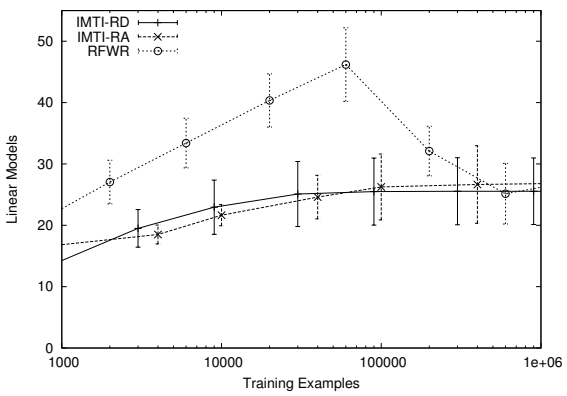


b. Number of local models.

Figure 27: Comparison of batch algorithms and 10-NN on the flight simulator.



a. Prediction error.



b. Number of local models.

Figure 28: Comparison of incremental algorithms on the flight simulator.

IMTI-RD and Batch-RD are able to use the Boolean 'on ground' indicator and the categorical flaps setting to form candidate splits (see section 4.5), however the alternative algorithms are unable to take these attributes into account.

5.5.1 Batch Algorithms

The Batch-RD stopping parameter $\delta_0 = 0.005$. SUPPORT uses the parameters MINDAT=50, f=0.7 and η=0.7. Figure 27 shows that Batch-RD learns a significantly better approximation than SUPPORT.

5.5.2 Incremental Algorithms

The stopping parameter $\delta_0 = 0.02$ for the RD splitting rule, and $\delta_0 = 0.01$ for the RA rule. The RFWR initial distance metric $\mathbf{D}_0 = 2.5\mathbf{I}$, the penalty $\gamma = 10^{-5}$ and the learning rates are 1000. Figure 28 shows that IMTI-RD learns a slightly more accurate and stable prediction than RFWR. IMTI-RA again takes longer to learn, and its final prediction is not quite as accurate. The linear models built by both IMTI algorithms and Batch-RD estimate a single state

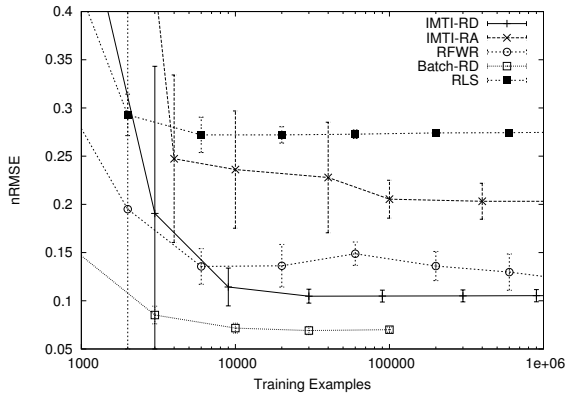


Figure 29: Learning the model of the plane on the ground.

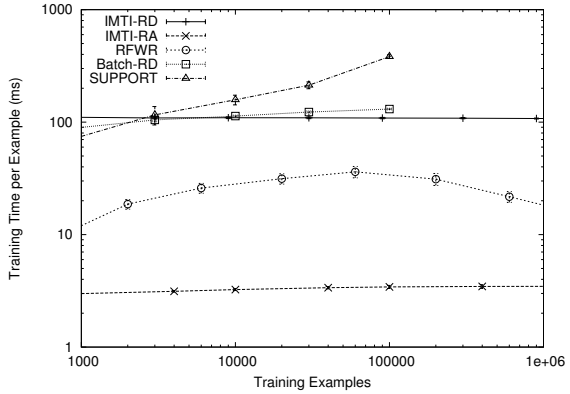


Figure 30: Performance on the flight simulator.

component and contain only 12 parameters (the Boolean and categorical attributes cannot contribute to the linear regressions), whereas the RFWR models predict all 8 components and contain 96 parameters each. Therefore the total number of parameters used by the model tree algorithms is far less than the number used by RFWR.

The aircraft clearly behaves very differently when it is touching the ground (e.g. the ailerons and elevator controls have little effect unless the speed is quite high). It would therefore be expected that the non-linear methods construct a significantly different model for this region of the state space than the model corresponding to the aircraft flying. To test this hypothesis the same experiment was conducted, but with a test set consisting only of examples with the plane on the ground. Figure 29 shows that the linear model does indeed perform badly on the ground, whereas RFWR, IMTI-RD and Batch-RD are able to learn an accurate model.

5.5.3 Performance

Figure 30 shows the poor performance of IMTI-RD on this higher dimensional problem and highlights the excellent scaling of IMTI-RA with dimensionality[2].

[2]Note the log-log scale, and that these results are obtained on a different platform and cannot be compared directly with previous performance results.

6 Discussion

For each algorithm various parameters are tuned in each domain to result in effective learning. RFWR has three such parameters; 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 size and number of linear models (it plays a similar role to the IMTI stopping parameter δ_0), and the learning rates affect how quickly the size and shape of the linear 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 linear models is observed.

In contrast the IMTI algorithms and Batch-RD require that only the stopping parameter is adjusted in each domain. This parameter directly controls the trade-off between final model size and prediction accuracy (see figure 6), and the algorithm is stable for any value of δ_0. In practice δ_0 proved easy to tune. An advantage of the tree representation in an incremental setting is that δ_0 can be reduced on-line if the approximation is not sufficiently accurate. The tree will grow from its leaves to form a more detailed model, and no re-building or internal restructuring is required.

SUPPORT uses the f parameter to control the final model size and prediction accuracy, while the threshold η appears to have more effect on the size of the trees built from smaller training sets. It proved straightforward to select f first and then tune η.

When representing dynamic models the model tree algorithms induce a separate tree for each state component. For example for the continuous pendulum (14), one model tree is built to represent $\dot{\theta}$ and one for $\ddot{\theta}$. With regressors θ, $\dot{\theta}$ and u, clearly $\dot{\theta}$ can be represented by a single linear model, while $\ddot{\theta}$ contains a sine wave component in the θ direction. IMTI therefore induces a single linear model 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, 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 are so effective in this setting.

When faced with large volumes of training data it becomes intractable to use batch or instance-based methods. On our platform Batch-RD, SUPPORT and 10-NN use too much time and memory to process more than 100,000 examples. Assuming that the number of local models in their piecewise linear approximations has stabilised, both IMTI and RFWR scale linearly with the number of examples and their memory usage is capped. However it can be seen that the RFWR training time is sensitive to the number of local models while the IMTI algorithms are not. This is because the RFWR training function updates all nearby models, whereas the IMTI training function only updates a single leaf. When IMTI processes a training example the RD splitting rule updates $\kappa(d-1)$ more models than the RA rule, and is therefore significantly slower. As expected the difference between the 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 then the RD rule is preferred, otherwise the RA rule may be more suitable.

One final point regarding the building of linear models in the leaves is that the recursive least squares algorithm also maintains the matrix $[\mathbf{X}^T\mathbf{X}]^{-1}$ (in the notation of section 2). Multiplying this by the noise variance (which is easily estimated) gives the covariance matrix of the linear model estimate $\hat{\boldsymbol{\theta}}_{\mathrm{LS}}$, and therefore a detailed measure of its uncertainty. This additional information allows confidence limits to be set and probabilistic reasoning to take place.

7 Conclusions and Future Work

This report describes and evaluates an algorithm that incrementally induces linear model trees. The algorithm can learn a more accurate approximation of an unknown function from fewer examples than other incremental methods, and in addition the induced model can contain less parameters.

The residual difference (RD) and residual analysis (RA) splitting rules are developed as two alternatives for incrementally growing model trees. The IMTI-RD algorithm is seen to accurately learn the non-linear dynamics in several domains. Its initial performance is good, and often comparable to its batch equivalent Batch-RD and the batch SUPPORT algorithm which both induce the same type of model tree. The RA rule suffers from slow initial learning and does not form a reasonable model tree in all domains. However it has a major computational advantage because it does not maintain a linear model on each side of every candidate split, and in the majority of domains builds a final model tree that is equally as effective.

In the IMTI algorithm the trade-off between the number of linear models built and the approximation error can easily be controlled using the single stopping parameter δ_0. Moreover there are no learning rates to be tuned, thus avoiding a major cause of instability in many gradient descent systems. Other benefits of the algorithm are that no initial knowledge regarding the size of the input domain is required, and no metric need be defined over this space (as opposed to RFWR which requires knowledge of the input domain to set the initial size of the local models, and instance-based methods that require a metric). In addition categorical attributes can be easily incorporated when using the RD splitting rule. Pruning, while having little impact in the smaller domains is seen to both limit the number of models and improve prediction accuracy in a more complex environment.

Having developed a method for rapidly learning non-linear models of dynamic environments, future work will concentrate on control. Nakanishi et al. (2002) 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

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 31).

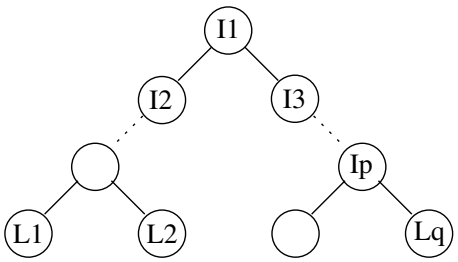


Figure 31: Labelling of tree nodes.

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 $\mathrm{RSS}_{0,\mathrm{Ij}}$ 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 $\mathrm{RSS}_{\mathrm{L}}$ 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 $\mathrm{RSS}_{\mathrm{L}}$ has the same distribution regardless. Following Chow (1960) H_0 can therefore be tested by the statistic

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

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.

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. *Proceedings of the European Working Session on Learning, Lecture Notes in Artificial Intelligence, 482*. 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*, 591–605.

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

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

Isaac, A., & Sammut, C. (2003). Goal-directed learning to fly. *Proceedings of the 20th International Conference of Machine Learning* (pp. 258–265).

Karalic, A. (1992). Employing linear regression in regression tree leaves. *Proceedings of the 10th European Conference on Artificial Intelligence* (pp. 440–441).

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

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., Appice, A., Bellino, A., Ceci, M., & Pallotta, D. (2001). Stepwise induction of model trees. *AI*IA 2001: Advances in Artificial Intelligence, Lecture Notes in Artificial Intelligence, 2175*. 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.

Nakanishi, J., Farrell, J., & Schaal, S. (2002). A locally weighted learning composite adaptive controller with structure adaptation. *IEEE International Conference on Intelligent Robots and Systems*.

Potts, D. (2004a). Incremental learning of linear model trees. To appear in: *Proceedings of the 21st International Conference on Machine Learning*.

Potts, D. (2004b). Fast incremental learning of linear model trees. To appear in: *Proceedings of the 8th Pacific Rim International Conference on Artificial Intelligence*.

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

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

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

Sicilano, R., & Mola, F. (1994). Modelling for recursive partitioning and variable selection. *Proceedings of Computational Statistics: COMPSTAT '94* (pp. 172–177).

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

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

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. *Proceedings of the 17th International Conference on Machine Learning* (pp. 1079–1086).

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