Dynamic trees for streaming and massive data.

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Data collection at a massive scale is becoming ubiquitous in a wide variety of settings, from vast offline databases to streaming real-time information. Learning algorithms deployed in such contexts must rely on single-pass inference, where the data history is never revisited. In streaming contexts, learning must also be temporally adaptive to remain up-to-date against unforeseen changes in the data generating mechanism. Although rapidly growing, the online Bayesian inference literature remains challenged by massive data and transient, evolving data streams. Non-parametric modelling techniques can prove particularly ill-suited, as the complexity of the model is allowed to increase with the sample size. In this work, we take steps to overcome these challenges by porting standard streaming techniques, like data discarding and downweighting, into a fully Bayesian framework via the use of informative priors and active learning heuristics. We showcase our methods by augmenting a modern non-parametric modelling framework, dynamic trees, and illustrate its performance on a number of practical examples. The end product is a powerful streaming regression and classification tool, whose performance compares favorably to the state-of-the-art.

1 Introduction

In online inference, the objective is to develop a set of update equations that incorporate novel information as it becomes available, without needing to revisit the data history. This results in model fitting algorithms whose space and time complexity remains constant as information accumulates, and can hence operate in streaming environments featuring continual data arrival, or navigate massive datasets sequentially. Such operational constraints are becoming increasingly imperative as modern data collection technology acquires data on a massive scale, and often in real-time.

In certain simple cases, online estimation without information loss is possible via exact recursive update formulae, e.g., via conjugate Bayesian updating (see Section 3.1). In parametric dynamic modelling, approximate samples from the filtering distribution for a variable of interest may be obtained online via sequential Monte Carlo (SMC) techniques, under quite general conditions. In [19], SMC is use in a non-parametric context, where a 'particle cloud' of trees are employed to track parsimonious regression and classification surfaces as data arrive sequentially. However, the resulting algorithm is not, strictly speaking, *online*, since local tree moves may revisit the partial data history depending on how the process evolves. This complication arises as an essential by-product of non-parametric modelling, wherein the complexity of the estimator is allowed to increase with the dataset size. Consequently, maintaining constant operational cost per timestep will necessarily entail discarding datapoints over time.

Our primary concern in this paper is managing the information loss entailed in data discarding. First, we propose datapoint *retirement* (Section 3), whereby discarded datapoints are partially 'remembered' through conjugate informative priors, updated sequentially. This technique is well-suited to trees, which combine non-parametric flexibility with simple parametric models with conjugate priors by partitioning. Nevertheless, forming new partitions in the tree still requires access to actual datapoints, and consequently data discarding comes at a cost of both information and degrees of freedom. We show that this can be managed, to a surprising extent, by the right retirement scheme even when discarding data randomly. We further show that borrowing active learning heuristics to prioritise points for retirement, i.e., *active retiring*, leads to better performance still.

An orthogonal concern in streaming data contexts is the need for temporal adaptivity when the concept being learned exhibits *drift*. This is where the data generating mechanism evolves over time in an unknown way. Recursive update formulae will generally require modification to acquire temporally adaptive properties. One common approach is the use of exponential downweighting, whereby the contribution of past datapoints to the algorithm is smoothly downweighted via the use of *forgetting factors*. In Section 5 we demonstrate how historical data retirement via suitably constructed informative priors can reproduce this effect in the non-parametric dynamic tree modelling context, while remaining fully online. Using a classic benchmark data set on electricity consumption, we show how this approach compares favorably against modern alternatives. The paper concludes in with a discussion in Section 6.

2 Dynamic Trees

Dynamic trees (DTs) [19] are a process–analog of Bayesian treed models [6, 7]. The model specification, and prior, are defined in a way is amenable to fast sequential inference by SMC, and that yields a predictive surface which organically increases in complexity as more data arrive. Here we review model specification and inference in turn. Software is available in the `dynaTree` package [10] for R on CRAN [18], which has been extended to cover the techniques described in this paper.

Model specification starts with the generative tree prior from [6]: a leaf node η may split with probability $p_{\mathrm{split}}(\mathcal{T}, \eta) = \alpha(1 + D_\eta)^{-\beta}$, where $\alpha, \beta > 0$, and D_η is the depth of η in the tree $\mathcal{T}$. This determines the joint prior via the probability that internal nodes $\mathcal{I}_\mathcal{T}$ have split and leaves $\mathcal{L}_\mathcal{T}$ have not: $\pi(\mathcal{T}) \propto \prod_{\eta \in \mathcal{I}_\mathcal{T}} p_{\mathrm{split}}(\mathcal{T}, \eta) \prod_{\eta \in L_\mathcal{T}} [1 - p_{\mathrm{split}}(\mathcal{T}, \eta)]$. In its original, static, formulation the specification is completed by employing independent sampling models at the tree leaves: $p(y_1, \ldots, y_n | \mathcal{T}, \mathbf{x}_1, \ldots, \mathbf{x}_n) = \prod_{\eta \in \mathcal{L}_\mathcal{T}} p(y^\eta | \mathcal{T}, \mathbf{x}^\eta)$. Sampling from the posterior proceeded by MCMC, via proposed for local changes to $\mathcal{T}$: so-called *grow*, *prune*, *change*, and *swap* "moves". Any data type/model may be used as long as the marginal likelihoods $p(y^\eta | \mathcal{T}, \mathbf{x}^\eta)$ are analytic, i.e., as long as their parameters can be integrated out. This is usually facilitated by fully conjugate, scale invariant, default (non-informative) priors.

In DTs the "moves" are embedded into the process, which describes how old trees mature into new ones when new data arrive. Suppose that $\mathcal{T}_{t-1}$ includes the recursive partitioning rules associated with $\mathbf{x}^{t-1}$, the set of covariates observed up-to time $t-1$. The DT process is defined by the set of trees $\{\mathcal{T}_t\}$ which are reachable from $\mathcal{T}_{t-1}$ when $\mathbf{x}_t$ arrives. These are available through one of three equally probable moves local to the leaf node $\eta(\mathbf{x}_t) \in \mathcal{T}_{t-1}$: *stay*, *prune* and *grow*. This makes the (prior) tree process data-dependent (but only on $\mathbf{x}$, not y). *Stay* moves keep the tree hierarchy unchanged, so that $\mathcal{T}_t = \mathcal{T}_{t-1}$. *Prune* moves remove $\eta(\mathbf{x}_t)$ as well as its sibling's subtree so that the parent of $\eta(\mathbf{x}_t)$ is a leaf in $\mathcal{T}_t$. *Grow* moves create a new pair of leaf nodes in $\mathcal{T}_t$ by randomly proposing to split on one of the data locations in $\eta(\mathbf{x}_t)$. Choosing uniformly over these three "moves" completes the specification of $p(\mathcal{T}_t | \mathcal{T}_{t-1}, \mathbf{x}^t)$. The new observation, y_t, completes a stochastic rule for determining how the update $\mathcal{T}_{t-1} \rightarrow \mathcal{T}_t$ shakes out, via $p(y^t | \mathcal{T}_t, \mathbf{x}^t)$ for each $\mathcal{T}_t \in \{\mathcal{T}_t\}$. As in the static tree model, the leaf marginal likelihood must be analytic. We therefore restrict our attention to *constant* and *linear* models for regression, and *multinomial* models for classification, which have this property [19].

As with many pragmatic modelling choices, the DT specification is "tuned" to be efficient under particular inferential mechanics, in this case SMC method of *particle learning* [5, PL]. At each iteration t, the discrete approximation to the tree posterior $\{\mathcal{T}_{t-1}^{(i)}\}_{i=1}^N$, based on N particles, is updated to $\{\mathcal{T}_t^{(i)}\}_{i=1}^N$ by first applying a *resample* step, and then a *propagate* step. The resample step resamples the particles according to their predictive probability for the next $(\mathbf{x}, y)$ pair, $w_i = p(y_t | \mathcal{T}_{t-1}^{(i)}, \mathbf{x}_t)$. Then the propagate step evolves each resampled particle following the process outlined above. Both steps are computationally efficient because they involve only local calculations (requiring only the subtrees of the parent of each $\eta^{(i)}(\mathbf{x})$), and because updating sufficient statistics at those subtrees involve standard recursions.

Despite local calculations, the particle approximation can shift great distances in posterior space after an update because the data governed by $\eta(\mathbf{x}_t)^{(i)}$ may differ greatly from one particle to another, and thus so may the weights w_i. Propagations, however, are truly local which (together with resampling) leads to an appealing division of labour, mimicking the behaviour of an ensemble method without explicitly maintaining one. As with all particle simulation methods, some Monte Carlo (MC) error will accumulate and, in practice, one must be careful to assess its effect. Nevertheless, DT out-of-sample performance compares favorably to other nonparametric methods, like Gaussian processes (GPs) regression and classification, but at a fraction of the computational cost [19].

3 Datapoint retirement

At time t, the DT algorithm of [19] would require access to the entire data history in order to compute $p(\mathcal{T}_t \mid \mathcal{T}_{t-1}, \mathbf{x}^t)$ for each particle. To enable online operation with constant memory requirements, the covariate pool, $\mathbf{x}^t = (\mathbf{x}_1, \ldots, \mathbf{x}_t)$ has to be of size constant with t. This can only be achieved via data selection, discussed in detail in Section 4. However, the analytic/parametric nature of DT leaves allows us to retain part of the discarded information in the form of informative priors, yielding in effect a *soft* implementation of data discarding, which we refer to as *datapoint retirement*. Updating the relevant leaf priors before discarding a datapoint proceeds via standard conjugate Bayesian updating, described in brief in Section 3.1. We explain the role of the resulting informative priors at the tree level in Section 3.2.

3.1 Conjugate informative priors at the leaf level

For simplicity, we first focus on a single leaf η, letting $(\mathbf{x}_n, y_n)_{n\in\mathcal{R}}$ denote the data retired from η so far. The leaf prior $\pi(\theta)$ is then given by the posterior of θ given the retired data:

$$\pi(\theta) =_{\text{df}} P\left(\theta \mid (\mathbf{x}_n, y_n)_{n\in\mathcal{R}}\right) \propto L\left(\theta; (\mathbf{x}_n, y_n)_{n\in\mathcal{R}}\right) \pi_0(\theta) \tag{1}$$

where $\pi_0(\theta)$ is a non-informative prior employed in the absence of retired data. Assume now that we wish to retire one more datapoint, $(\mathbf{x}_m, y_m)$. We may recursively update the prior as follows:

$$\pi^{(\text{new})}(\theta) =_{\text{df}} P\left(\theta \mid (\mathbf{x}_n, y_n)_{n\in\mathcal{R}\cup\{m\}}\right) \propto L(\theta; \mathbf{x}_m, y_m) P\left(\theta \mid (\mathbf{x}_n, y_n)_{n\in\mathcal{R}}\right) \tag{2}$$

As shown below, the calculation in (2) is tractable whenever conjugate priors are employed.

Consider first the linear regression model, $\mathbf{y} \sim N(\mathbf{X}\beta, \sigma^2\mathbf{I})$, where $\mathbf{y} = (y_n)_{n\in\mathcal{R}}$ is the retired response data, and $\mathbf{X}$ the retired *augmented* design matrix, i.e., consisting of augmented covariate vectors $[1, \mathbf{x}_n']'$, so that β_1 represents an intercept. With $\pi_0(\beta, \sigma^2) \propto \frac{1}{\sigma^2}$, we obtain:

$$\pi_t(\beta, \sigma^2) =_{\text{df}} P(\beta, \sigma^2 \mid \mathbf{y}, \mathbf{X}) = \text{NIG}(\nu/2, s\nu/2, \beta, \mathbf{A}) \tag{3}$$

where NIG stands for Normal-Inverse-Gamma, and assuming $\mathbf{X}'\mathbf{X}$ is invertible,

$$\mathbf{A} = \left(\mathbf{X}'\mathbf{X}\right)^{-1}, \ \mathbf{R} = \mathbf{X}'\mathbf{y}, \ r = \mathbf{y}'\mathbf{y}, \ \nu = n - p, \ \beta = \mathbf{A}\mathbf{R}, \ s^2 = \frac{1}{\nu}\left(r - \beta'\mathbf{A}\beta\right) \tag{4}$$

Although the retired data $(y_n, \mathbf{x}_n)_{n\in\mathcal{R}}$ are unavailable, we can afford to keep in memory the values of the above statistics, as their dimension does not grow with the size of $\mathcal{R}$. If we now wish to retire an additional datapoint $(y_m, \mathbf{x}_m)$, we perform the following updates:

$$\left(\mathbf{A}^{(\text{new})}\right)^{-1} = \mathbf{A}^{-1} + X_m'X_m, \ \mathbf{R}^{(\text{new})} = \mathbf{R} + \mathbf{X}_m'\mathbf{y}_m, \ s^{(\text{new})} = s + y_m'y_m, \ \nu^{(\text{new})} = \nu + 1. \tag{5}$$

Having presented in detail the linear model, we now present in brief the constant and multinomial cases, referring to [16] for more details. The constant leaf model may be obtained as a special case of the above, where $\mathbf{x}^\star = 1$, $\mathbf{A} := 1/\nu$ and $\beta = \mu$, i.e., the intercept now represents the mean of y. For the multinomial model, the discarded response values $(y_n)_{n\in\mathcal{R}}$ may be represented as indicator vectors $(\mathbf{z}_n)_{n\in\mathcal{R}}$, where $z_{jn} = \mathbf{1}(y_n = j)$. The natural conjugate here is the *Dirichlet* $D(\mathbf{a})$. The hyperparameter vector $\mathbf{a}$ may be interpreted as counts, and is updated in the obvious manner, namely $\mathbf{a}^{(new)} = \mathbf{a} + z_m$ where $z_{jm} = \mathbf{1}(y_m = j)$. We use $\mathbf{a}_0 = (1, 1, \ldots, 1)$, and also find it useful to employ a normalised representation of $\mathbf{a}$, given by $\mathbf{t} = \mathbf{a}/\sum_j a_j$.

3.2 Employing informative priors at the tree level

Intuitively, the DT with retirement manages two types of information: a non-parametric memory comprising an active data pool of constant size w, which forms the leaf likelihoods; and a parametric memory consisting of possibly informative leaf priors. To give context, the original DT differed in featuring an active data pool of increasing size, and non-informative priors throughout.

The DT with retirement algorithm will, at time $t+1$, add the $(t+1)$th datapoint to the active pool, and update the model by SMC exactly as explained in Section 2. Once t exceeds w, the algorithm will also select some datapoint, $(\mathbf{x}_r, y_r)$, to retire from the active pool (see Section 4 for selection criteria). Following retirement, the associated leaf prior for $\eta(\mathbf{x}_r)^{(i)}$ for each particle $i = 1, \ldots, N$ will be updated to incorporate the information present in $(\mathbf{x}_r, y_r)$. It is important to observe that at this level, data retirement performs a shifting of information from the likelihood part of the posterior to the prior that preserves, exactly, the posterior predictive distribution and the value of the marginal likelihood calculation both locally at each node $\eta(x)^{(i)}$, as well as globally for each tree $\mathcal{T}^{(i)}$. Moreover, this predictive distribution is preserved for any future updates involving data points that either belong to a different node, or belong to $\eta(\mathbf{x})^{(i)}$ which stochastically chose a *stay* move. In brief, so long as a leaf remains fixed, data retirement leaves the leaf model unaffected.

The retiring mechanism, which dictates how informative priors can salvage discarded likelihood information, suggests a method for splitting and combining that information in 'grow' and 'prune' moves. Following a 'prune' move, retired datapoints from the pruned leaves are pooled together to form the new leaf prior in the obvious additive manner implied by conjugate updating. Note that this does not require access to the actual retired datapoints. Following a 'grow' move, we let both novel leaves inherit the parent prior, but split its strength $\nu^{(P)}$ between the two at proportions equal to the active data proportions in the children. For instance, if the two newly populated leaves feature 20 and 80 active datapoints respectively, the strengths of their respective priors $\nu^{(L)}$ and $\nu^{(R)}$ will satisfy $\nu^{(L)} = 0.2\nu^{(P)}$ and $\nu^{(R)} = 0.8\nu^{(P)}$. This preserves the total strength of retired information, as well as the balance between active data and parametric memory.

4 Active discarding

Retirement softens the blow of data discarding by moving leaf sufficient statistics into the leaf priors. Still, in leaves that are touched by future data, the degrees of freedom lost to retired points are missed, particularly for *grow* which relies on the active data pool for possible splitting locations. Consequently, it matters which data points are chosen for retirement. Therefore, we want to retire datapoints that will be of "less" use to the model going forward. In the case of a drifting concepts, retiring historically may be sensible. We address this in Section 5. Here we consider static concepts, or in other words i.i.d. data. We formulate the choice of which active data points to retire as an *active retiring* (AR) problem, i.e., borrowing (and reversing) techniques from the *active learning* (AL) literature. We treat the regression and classification models separately, as they require different AR techniques. We shall argue that in both cases AR is, in fact, easier than AL since DTs enable thrifty analytic calculations not previously possible, which are easily updated within the SMC.

4.1 Active discarding for regression

Active learning (AL) procedures are sequential decision heuristics for choosing data to add to the design, usually with the aim of minimising prediction error. Two common AL heuristics are active learning MacKay [15, ALM] and active learning Cohn [8, ALC]. They were popularised in the modern nonparametric regression literature [17] using GPs, and subsequently ported to DTs [19]. An ALM scheme selects new inputs $\mathbf{x}^\star$ with maximum variance for $y(\mathbf{x}^\star)$, whereas ALC chooses $\mathbf{x}^\star$ to maximise the expected reduction in predictive variance averaged over the input space. Both approximate maximum expected information designs in certain cases. ALC is computationally more demanding than ALM, requiring an integral over a set of reference locations that can be expensive to approximate numerically for most models. But it leads to better exploration when used with nonstationary models like DTs [19]. Both are sensitive to the choice of (and density of) a search grid over which the variance statistics are evaluated.

Our first simplification when porting AL to AR is to recognise that no grids are needed. We focus on the ALC statistic here because ALM is trivial, and we aim to show that the integrals are very tractable with DTs. The program is to evaluate the ALC statistic at each active data location, and choose the smallest one for discarding — AL would instead choose the largest one to augment the design. We focus on the linear case, as the constant case may be derived as a special case. We have:

$$\Delta\sigma^2_{\mathbf{x}}(\mathbf{z}|\mathcal{T}) = \Delta\sigma^2_{\mathbf{x}}(\mathbf{z}|\eta) \equiv \sigma^2(\mathbf{z}|\eta) - \sigma^2_x(\mathbf{z}|\eta) = \frac{s^2_\eta - \mathcal{R}_\eta}{|\eta| - m - 3} \times \frac{\left(\frac{1}{|\eta|} + \mathbf{z}'\mathcal{G}_\eta^{-1}\mathbf{x}\right)^2}{1 + \frac{1}{|\eta|} + \mathbf{x}'\mathcal{G}_\eta^{-1}\mathbf{x}},$$

when *both* $\mathbf{x}$ and $\mathbf{z}$ are in $\eta \in \mathcal{L}_\mathcal{T}$, and zero otherwise. The s^2_η is the leaf sum of squares, $|\eta|$ is the number of data points in the leaf, and $\mathcal{G}_\eta$ is the Grahm matrix for η for the active *and* retired data [19]. Integrating over $\mathbf{z}$ gives:

$$\Delta\sigma^2(\mathbf{x}) = \int_{\mathbb{R}^d} \Delta\sigma^2_{\mathbf{x}}(\mathbf{z})\, d\mathbf{z} = \frac{s^2_\eta - \mathcal{R}_\eta}{(|\eta| - m - 3)(1 + \frac{1}{|\eta|} + \mathbf{x}'\mathcal{G}_\eta^{-1}\mathbf{x})} \times \int_\eta \left(\frac{1}{|\eta|} + \mathbf{z}'\mathcal{G}_\eta^{-1}\mathbf{x}\right)^2 d\mathbf{y}.$$

The integral that remains, over the rectangular region η, is tedious to write out, but easy to implement. It is standard polynomial integration with a trivial $O(m^2)$ implementation:

$$\int_{a_1}^{b_1} \cdots \int_{a_m}^{b_m} \left(c + \sum_{i=1}^m \tilde{z}_i x_i\right)^2 dz_1 \cdots dz_m = A_\eta c^2 + c\sum_i \left(\prod_{k\neq i}(b_k - a_k)\right) x_i(b_i^2 - a_i^2)$$
$$+ \sum_i \left(\prod_{k\neq i}(b_k - a_k)\right) \frac{x_i^2}{3}(b_i^3 - a_i^3) + \sum_i \sum_{j<i} \left(\prod_{k\neq i,j}(b_k - a_k)\right) \frac{x_i x_j}{2}(b_i^2 - a_i^2)(b_j^2 - a_j^2),$$

where the m-rectangle η is being described by $\{(a_i, b_i)\}^m$, $\tilde{\mathbf{z}} = \mathbf{z}'\mathcal{G}_\eta^{-1}$, and $c = 1/|\eta|$. A numerical version via sums using R reference locations $\mathbf{z}$ is possible, but this requires $O(Rm)$ computation and R must grow exponentially in m for a reasonable approximation. Observe that the rectangular leaf regions generated by the trees is key. In the case of other partition models (like Voronoi tessellation models), this analytical integration would not be possible.

In repeated applications of ALC for AR, the active points that remain tend shuffle themselves so that they cluster near the boundaries of high posterior partitioning boundaries. The number of such locations depends crucially on the number of active data points allowed, w. As an illustration, consider

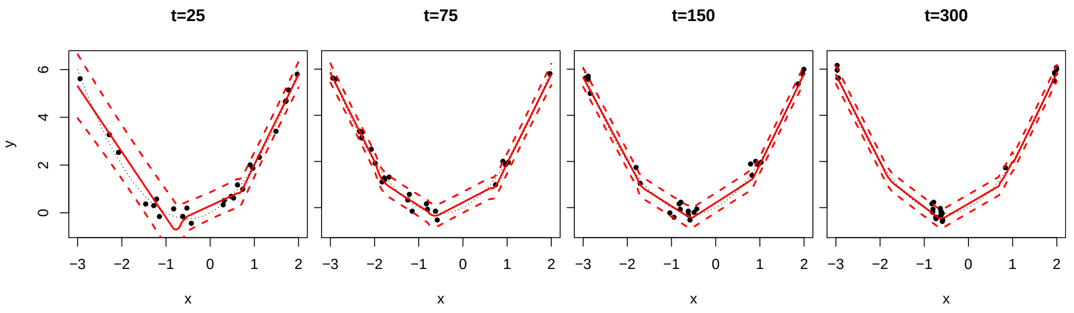


Figure 1: Snapshots of active data (25 points) and predictive surfaces spanning 275 retirement/updating rounds; $Y(x) = x + x^2 + \varepsilon$, $\varepsilon \sim \mathcal{N}(0, 1)$.

the simple example where the response is a parabolic function, which must be learned sequentially via x-data sampled uniformly in $(-3, 2)$, with $w = 25$. The initial 25, before any retiring, are shown in the first panel. Each updating round then proceeds with one retirement followed by one new pair, and subsequent SMC update. Notice how the active points shift towards likely partition boundaries. Since the implementation requires at least five points in each leaf, seeing four regimes emerge is perhaps not surprising. By $t = 150$, the third pane, the ability to learn about the mean with just 25 degrees of freedom is saturated, but it is possible to improve on the variance (shown as errorbars), which are indeed smaller in the final $t = 300$ pane.

4.2 Active discarding for classification

For classification, predictive entropy is an obvious AL heuristic. Given a predictive surface comprised of probabilities $p_\ell(\mathbf{x})$ for each class ℓ, from DTs or otherwise, the predictive entropy at $\mathbf{x}$ is $-\sum_\ell p_\ell(\mathbf{x}) \log p_\ell(\mathbf{x})$. Entropy can be an optimal method for measuring predictive uncertainty, but that does not mean it is good for AL. Many authors (e.g., [13]) have observed that it can be too greedy. Although entropy is low far from class boundaries and higher near to them (a good thing) it can be higher still near the best explored boundaries. In other words, the higher the certainty about a boundary, the higher the entropy there, making it poor for exploring the full boundary set.

Several, largely unsatisfactory, remedies have been suggested in the literature. Fortunately, no remedy is required in the AR analog, which focuses on the lowest entropy active data, of which there are a finite set. The retained (non-retired) points will be no closer to the boundary than they already are, and the discarded ones no further into the interior, where they can be safely subsumed into the prior. Their spacing and shifting behaviour in sequential application is quite similar to discarding by ALC for regression, and so we do not illustrate it here.

4.3 Fast local updates of active discarding statistics with trees

The divide and conquer nature of trees—whose posterior distribution is approximated by thrifty, local, particle updates—allows AR statistics to be updated cheaply too. If each leaf node stores its own AR statistics, then rather than re-calculating ALC or entropy for every active input, after each update step, it suffices to update only the ones in leaf node(s) which have been modified. Any recalculated statistics can then be subsumed into the global, particle averaged, version (after first subtracting off the old values). Note that no updates to the AD statistics are needed when a point is retired since the predictive distributions are unchanged.

When a new datapoint $(\mathbf{x}, y)$ arrives, the posterior undergoes two types of changes: resample then propagate. In the resample step the discrete particle distribution changes, although the trees therein do not change. Therefore, each discarded particle must have its AD statistics (stored at the leaves) subtracted from the full particle tally. Then each correspondingly duplicated particle can have its AD statistics added in. No new integrals (for ALC) or entropy calculations (for classification) are needed. In the propagate step, each particle undergoes a change local to $\eta(\mathbf{x})^{(i)} \in \mathcal{T}_t^{(i)}$. This requires first calculating the AR statistic for the new $(\mathbf{x}, y)$ for each $\eta^{(i)}(\mathbf{x})$, before the dynamic update occurs, and then swapping it into the particle average. New integrations, etc., need evaluating here. Then, each non-*stay* dynamic update triggers swap of the old AR statistics in $\eta^{(i)}(\mathbf{x})$ for freshly re-calculated ones from the leaf node(s) in $\mathcal{T}_{t+1}^{(i)}$. The total computational cost is in $O(m^2 N)$ for incorporating $(\mathbf{x}, y)$ into N particles, plus $O(m^2 \sum_{i=1}^N |\eta^{(i)}(x)|)$ to update the leaves[1].

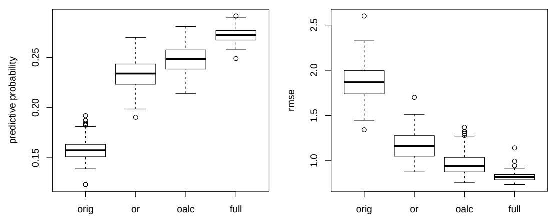
4.4 Empirical results

Here we explore the benefit of AR over simpler heuristics, like random discarding and subsetted data estimators, by making predictive comparisons on benchmark regression and classification data. The problems are of modest size in order to enable a comparison to full-data versions. It is worth noting that DTs main competitors, like GP-based regression and classification models, simply cannot be used on data sets of the size we use here.

We first consider data originally used to illustrate multivariate adaptive regression splines (MARS) [9], and then to demonstrate the competitiveness of DTs relative to modern (batch) nonparametric models [19]. The response is $10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5$ plus $\mathcal{N}(0,1)$ additive error. Inputs $\mathbf{x}$ are random in $[0,1]^5$. We considered four estimators: one based on 200 pairs (orig), one based 1800 more for 2000 total (full), and two online versions [which saw the same stream of pairs] using either random (orand) or ALC (oalc) retiring to keep the total active data set limited to $w = 200$. We repeated the experiment 100 times in a MC fashion, each with new random training sets, and random testing sets of size 1000. $N = 1000$ particles and a linear leaf model were used throughout. Similar results were obtained for the constant model. From Figure 2 we can see that

[1]One might imagine a thriftier, but harder to implement version, which waits until the end to calculate the active discarding statistic for the new point (x, y). But it would have the same computational order.

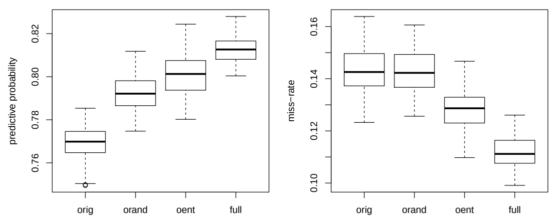
random retiring is better than subsetting , but retiring by ALC is even better, and can be nearly as good as the full-data estimator. In fact, it may be surprising to note that "oalc" was the *best* predictor 6% and 12% of the time by average predictive density and RMSE, respectively. The average time used by each estimator was approximately 1, 37, 52, and 78 seconds, respectively. So random retiring on this modesly-sized problem is 2-times faster than using the full data. ALC costs about 19% extra, time-wise, but leads to about a 14% reduction in RMSE relative to the full estimator. We note that in much larger problems the gap between the online and full estimators can widen considerably. The time-demands of the full estimator grow roughly as $t \log t$, whereas the online versions stay constant.



	average predictive density		
	mean	5%	95%
orig	0.15779	0.14019	0.18247
orand	0.23284	0.20931	0.25143
oalc	0.24825	0.22111	0.27317
full	0.27226	0.26117	0.28321
	rmse		
orig	1.87061	1.52388	2.20422
orand	1.17517	0.95645	1.41653
oalc	0.98058	0.78805	1.28089
full	0.82610	0.75729	0.90864

Figure 2: Friedman data comparisons by average posterior predictive density (higher is better) and RMSE (lower is better).

Now consider the Spambase data set, from the UCI Machine Learning Repository [2]. The data contains binary classifications of 4601 emails based on 57 attributes (predictors). We do a similar experiment to the Friedman/regression example, above, except with classification leaves and 5-fold CV to create training and testing sets. This was repeated twenty times, randomly, giving 100 sets total. We considered four estimators: one based on 1/10 of the training fold (ORIG), one based on the full fold (FULL), and two online versions trained on the same stream(s) using either random (ORAND) or entropy (OENT) retiring to keep the total active data set limited to 1/10 of the full set. Figure 3 tells a similar story to the Friedman experiment: random discarding is better than subsetting, but discarding by entropy is even better, and can be nearly as good as the full-data estimator. Entropy retiring resulted the best predictor 15% and 5% of the time by predictive probability and misclassification rate, respectively. Observe that while ORAND predicts the label about as well as ORIG, it is much better at accurately capturing the full predictive distribution.



	average predictive probability		
	mean	5%	95%
ORIG	0.76917	0.75198	0.78233
ORAND	0.79216	0.77807	0.80536
OENT	0.80140	0.78760	0.81720
FULL	0.81260	0.80415	0.82317
	misclassification rate		
orig	0.14343	0.13021	0.16138
orand	0.14261	0.12903	0.15545
oent	0.12761	0.11433	0.13870
full	0.11170	0.10151	0.12089

Figure 3: Spam data comparisons by average posterior predictive probability (higher is better) and misclassification rate (lower is better) on the testing set(s).

5 Temporal adaptivity using forgetting factors

Data retirement causes accumulation of historical information at the leaf priors, which may eventually overpower the likelihood of active datapoints. As explained earlier, this is natural in an i.i.d. setting, but may cause performance deterioration in streaming contexts where the data generating

Method	H	H0.85	H0.8	E0.85	P-ALR	OLC-TF
CCR	0.831	0.913	0.925	0.841	0.851	0.831

Table 1: Correct Classification Rate (CCR) for ELEC2 dataset.

mechanism is likely to be subject to unforeseen disturbances. To promote responsiveness, we may *exponentially downweight* the retired data history $\mathcal{R}$ when retiring an additional point y_m:

$$\pi_\lambda^{(\text{new})}(\theta) \propto L(\theta \mid y_m) L^\lambda(\theta; (y_n, \mathbf{x}_n)_{n\in\mathcal{R}})\pi_0(\theta)$$

By inspection we observe that for $\lambda = 1$, conjugate Bayesian updating is recovered, and for $\lambda = 0$, the retired history is disregarded altogether, effectively 'resetting' the prior. For $\lambda \in (0, 1)$, retiring a datapoint will move our prior in the direction of the recently retired datapoint, diminishing the influence of older retired data. For the leaf models entertained in this paper, a recursive application of this principle, with $\lambda \in (0, 1)$, modifies only slightly the conjugate updates of Section 3, as follows. For the linear and constant models, we have $\left(\mathbf{A}^{(\text{new})}\right)^{-1} = \lambda\mathbf{A}^{-1} + X_m' X_m$, $\mathbf{R}^{(\text{new})} = \lambda\mathbf{R} + \mathbf{X}_m'\mathbf{y}_m$, $s^{(\text{new})} = \lambda s + y_m' y_m$, and $\nu^{(\text{new})} = \lambda\nu + 1$, whereas for the multinomial, we get $\mathbf{a}^{(new)} = \mathbf{a} + z_m$.

In [12], this family of priors is shown to satisfy desirable information-theoretic optimality properties. Exponential downweighting as a means of enabling temporal adaptivity also has a long tradition in non-stationary signal processing [11] where λ is often referred to as a *forgetting factor*. It is important to note that for $\lambda < 1$, κ and ν will be bounded above by their limiting value $\frac{1}{1-\lambda}$. This results in informative priors of *bounded strength*, which enables the model to remain responsive.

We proceed to test the performance of the DT with forgetting using a real streaming classification datasets, and compare to two state-of-the-art streaming classifiers. The dataset we consider, denoted by $ELEC2$, holds information for the Australian New South Wales Electricity Market, and has become a benchmark of sorts in the streaming classification literature (e.g., [3, 14]), as it has shown evidence of drifting data distributions. It comprises 27552 instances, each referring to a period of 30 minutes. The class label identifies the price change related to a moving average of the last 24 hours, and the four covariates capture aspects of electricity demand and supply. We compare to two competing methods from the streaming classification literature: a perceptron featuring adaptive learning rates (P-ALR) proposed in [14], and an online logistic classifier with tuned forgetting (OLC-TF), originally designed for the massive data setting [4] and adapted for the streaming context in [1]. Results are presented in Table 5. DT with historical retirement without forgetting factors (H) is already competitive to state-of-the-art methodology, suggesting that the non-parametric, dynamic nature of the model is capable of learning drifting concepts without the need for exponential forgetting. Nevertheless, DT with $\lambda = 0.8$, denoted by H0.8, significantly outperforms all other methods. A higher value of λ (H0.85) performs a little worse in this dataset, raising the question of automatic tuning of the forgetting factor as a point for future work. We note that retirement via entropy (E0.85) has an adverse effect on performance in this context. This is perhaps natural, given that active discarding keeps the most surprising datapoints which, in the presence of drift, are likely to include obsolete data poorly explained by the model.

6 Conclusion

In this work, we strive to fully utilise the potential of Bayesian machinery in the context of streaming non-parametrics. We propose data retirement via conjugate Bayesian updating in the context of SMC inference for a DT model, preserving non-parametric flexibility while enabling constant memory online operation. Second, the availability of tractable predictive distributions allows us to devise computationally efficient active retirement heuristics, hence maintaining a fixed budget of highly informative datapoints. Both features minimise information loss incurred by single-pass processing. Finally, we deploy informative power priors to enable temporal adaptivity. This results in a novel, powerful algorithmic scheme for non-parametric regression and classification tailored to the massive and streaming data contexts. As future work, we intend to pursue techniques for automatic tuning of forgetting factors in streaming contexts, and their interplay with active retirement heuristics.

References

[1] C. Anagnostopoulos, D.K. Tasoulis, N.M. Adams, and D.J. Hand. Temporally adaptive estimation of logistic classifiers on data streams. *Advances in Data Analysis and Classification*, 3(3):243–261, 2009.

[2] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.

[3] M. Baena-Garcia, J. del Campo-Avila, R. Fidalgo, A. Bifet, R. Gavalda, and R. Morales-Bueno. Early drift detection method. In *ECML PKDD 2006 Workshop on Knowledge Discovery from Data Streams*, pages 77–86, 2006.

[4] D Balakrishnan, S; Madigan. Algorithms for sparse linear classifiers in the massive data setting. *Journal of Machine Learning Research*, 9:313–337, 2008.

[5] Carlos M. Carvalho, Michael Johannes, Hedibert F. Lopes, and Nicholas G. Polson. Particle learning and smoothing. *Statistical Science*, 25:88–106, 2010.

[6] H.A. Chipman, E.I. George, and R.E. McCulloch. Bayesian CART model search (with discussion). *Journal of the American Statistical Association*, 93:935–960, 1998.

[7] H.A. Chipman, E.I. George, and R.E. McCulloch. Bayesian treed models. *Machine Learning*, 48:303–324, 2002.

[8] D. A. Cohn. Neural network exploration using optimal experimental design. In *Advances in Neural Information Processing Systems*, volume 6(9), pages 679–686. Morgan Kaufmann Publishers, 1996.

[9] J. H. Friedman. Multivariate adaptive regression splines. *Annals of Statistics*, 19, No. 1:1–67, March 1991.

[10] Robert B. Gramacy and Matt A. Taddy. `dynaTree`*: Dynamic trees for learning and design*, 2011. R package version 2.0.

[11] S. Haykin. *Adaptive Filter Theory*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1996.

[12] J.G. Ibrahim, M.H. Chen, and D. Sinha. On optimality properties of the power prior. *Journal of the American Statistical Association*, 98(461):204–213, 2003.

[13] A.J. Joshi, F. Porikli, and N. Papanikolopoulos. Multi-class active learning for image classification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009. to appear.

[14] L.I. Kuncheva and C.O. Plumpton. Adaptive learning rate for online linear discriminant classifiers. *SSPR and SPR 2008, Lecture Notes in Computer Science (LNCS)*, 5342:510–519, 2008.

[15] D. J. C. MacKay. Information–based objective functions for active data selection. *Neural Computation*, 4(4):589–603, 1992.

[16] A. O'Hagan and J. Forster, editors. *Kendall's Advanced Theory of Statistics, Volume 2B, Bayesian Inference*. Arnold Publishers, 2004.

[17] S. Seo, M. Wallat, T. Graepel, and K. Obermayer. Gaussian process regression: Active data selection and test point rejection. In *Proceedings of the International Joint Conference on Neural Networks*, volume III, pages 241–246. IEEE, July 2000.

[18] R Development Core Team. R*: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2010. ISBN 3-900051-07-0.

[19] M.A. Taddy, R.B. Gramacy, and N.G. Polson. Dynamic trees for learning and design. *Journal of the American Statistical Association*, 106(493):109–123, 2011.