Top-down particle filtering for Bayesian decision trees

Balaji Lakshminarayanan BALAJI@GATSBY.UCL.AC.UK
Gatsby Unit, CSML, University College London

Daniel M. Roy D.ROY@ENG.CAM.AC.UK
University of Cambridge

Yee Whye Teh Y.W.TEH@STATS.OX.AC.UK
Department of Statistics, University of Oxford

Abstract

Decision tree learning is a popular approach for classification and regression in machine learning and statistics, and Bayesian formulations—which introduce a prior distribution over decision trees, and formulate learning as posterior inference given data—have been shown to produce competitive performance. Unlike classic decision tree learning algorithms like ID3, C4.5 and CART, which work in a top-down manner, existing Bayesian algorithms produce an approximation to the posterior distribution by evolving a *complete* tree (or collection thereof) iteratively via local Monte Carlo modifications to the structure of the tree, e.g., using Markov chain Monte Carlo (MCMC). We present a sequential Monte Carlo (SMC) algorithm that instead works in a top-down manner, mimicking the behavior and speed of classic algorithms. We demonstrate empirically that our approach delivers accuracy comparable to the most popular MCMC method, but operates more than an order of magnitude faster, and thus represents a better computation-accuracy tradeoff.

1. Introduction

Decision tree learning algorithms are widely used across statistics and machine learning, and often deliver near state-of-the-art performance despite their simplicity. Decision trees represent predictive models from an input space, typically $\mathbb{R}^D$, to an output space of labels, and work by specifying a hierarchical partition of the input space into blocks. Within each block of the input space, a simple model predicts labels.

In classical decision tree learning, a decision tree (or collection thereof) is learned in a greedy, top-down manner from the examples. Examples of classical approaches that learn single trees include ID3 (Quinlan, 1986), C4.5 (Quinlan, 1993) and CART (Breiman et al., 1984), while methods that learn combinations of decisions trees include boosted decision trees (Friedman, 2001), Random Forests (Breiman, 2001), and many others.

Bayesian decision tree methods, like those first proposed by Buntine (1992), Chipman et al. (1998), Denison et al. (1998), and Chipman & McCulloch (2000), and more recently revisited by Wu et al. (2007), Taddy et al. (2011) and Anagnostopoulos & Gramacy (2012), cast the problem of decision tree learning into the framework of Bayesian inference. In particular, Bayesian approaches start by placing a prior distribution on the decision tree itself. To complete the specification of the model, it is common to associate each leaf node with a parameter indexing a family of likelihoods, e.g., the means of Gaussians or Bernoullis. The labels are then assumed to be conditionally independent draws from their respective likelihoods. The Bayesian approach has a number of useful properties: e.g., the posterior distribution on the decision tree can be interpreted as reflecting residual uncertainty and can be used to produce point and interval estimates.

On the other hand, exact posterior computation is typically infeasible and so existing approaches use approximate methods such as Markov chain Monte Carlo (MCMC) in the batch setting. Roughly speaking, these algorithms iteratively improve a complete decision tree by making a long sequence of random, local modifications, each biased towards tree structures with higher posterior probability. These algorithms stand in marked contrast with classical decision tree learning algorithms like ID3 and C4.5, which rapidly build a de-

Proceedings of the 30^{th} International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

cision tree for a data set in a top-down greedy fashion guided by heuristics. Given the success of these methods, one might ask whether they could be adapted to work in the Bayesian framework.

In this article, we present such an adaptation, proposing a sequential Monte Carlo (SMC) method for approximate inference in Bayesian decision trees that works by sampling a collection of trees in a top-down manner like ID3 and C4.5. Unlike classical methods, there is no pruning stage after the top-down learning stage to prevent over-fitting, as the prior combines with the likelihood to automatically cut short the growth of the trees, and resampling focuses attention on those trees that better fit the data. In the end, the algorithm produces a collection of sampled trees that approximate the posterior distribution. While both existing MCMC algorithms and our novel SMC algorithm produce approximations to the posterior that are exact in the limit, we show empirically that our algorithms run more than an order of magnitude faster than existing methods while delivering the same predictive performance.

The article is organized as follows: we begin by describing the Bayesian decision tree model precisely in Section 2, and then describe the SMC algorithm in detail in Section 3. Through a series of empirical tests, we demonstrate in Section 4 that this approach is fast and produces good approximations. We conclude in Section 5 with a discussion comparing this approach with existing ones in the Bayesian setting, and point towards future avenues.

2. Model and notation

In this section, we present the decision tree model for the distribution of the labels $Y = \{y_n\}_{n=1}^N$ corresponding to input vectors $X = \{x_n\}_{n=1}^N$, $x_n \in \mathbb{R}^D$. The assumption is that the probabilistic mapping from input vectors to their labels is mediated by a latent decision tree $\mathcal{T}$ that serves to partition the input space into axis-aligned blocks. Each block is then associated with a parameter that determines the distribution of the labels of the input vectors falling in that block.

A rooted, strictly binary tree T is a finite tree with a single root, denoted by the empty string ϵ, where each internal node p except the root has exactly two children, called the left child $p0$ and the right child $p1$. Denote the leaves of T (those nodes without children) by $\partial\mathsf{T}$. Each node of the tree $p \in \mathsf{T}$ is associated with a block $B(p) \subset \mathbb{R}^D$ of the input space as follows: At the root we have $B(\epsilon) = \mathbb{R}^D$, while each internal node $p \in \mathsf{T} \setminus \partial\mathsf{T}$ "cuts" its block into two halves, with $\kappa(p) \in \{1, \ldots, D\}$ denoting the dimension of the cut, and $\tau(p)$ denoting the location of the cut, so that

$$\begin{aligned} B(p0) &= B(p) \cap \{\boldsymbol{z} \in \mathbb{R}^D : z_{\kappa(p)} \leq \tau(p)\} \text{ and} \\ B(p1) &= B(p) \cap \{\boldsymbol{z} \in \mathbb{R}^D : z_{\kappa(p)} > \tau(p)\}. \end{aligned} \tag{1}$$

We call the tuple $\mathcal{T} = (\mathsf{T}, \kappa, \tau)$ the decision tree. (See Figure 1 for more intuition on the representation and notation of decision trees.) Note that the blocks associated with the leaves of the tree partition $\mathbb{R}^D$. It will be convenient to write $N(p)$ for the set of data point indices $n \in \{1, \ldots, N\}$ such that $x_n \in B(p)$. For every subset $A \subseteq \{1, \ldots, N\}$, let $Y_A := \{y_n : n \in A\}$ and similarly for X_A, so that $X_{N(p)}$ are the input vectors in block $B(p)$ and $Y_{N(p)}$ are their labels. Note that both $B(p)$ and $N(p)$ depend on $\mathcal{T}$, although we have chosen to elide this dependence for notational simplicity.

Conditioned on the examples X, we assume that the joint density $f(Y, \mathcal{T} \mid X)$ of the labels Y and the latent decision tree $\mathcal{T}$ factorizes as follows:

$$\begin{aligned} f(Y, \mathcal{T} \mid X) &= h(\mathcal{T} \mid X)\, g(Y \mid \mathcal{T}, X) \\ &= h(\mathcal{T} \mid X) \textstyle\prod_{p \in \partial\mathsf{T}} \ell(Y_{N(p)} | X_{N(p)}) \end{aligned} \tag{2}$$

where ℓ denotes a likelihood, defined below.

In this paper, we focus on the case of categorical labels taking values in the set $\{1, \ldots, K\}$. It is natural to take ℓ to be the Dirichlet-Multinomial likelihood, corresponding to the data being conditionally i.i.d. draws from a multinomial distribution on $\{1, \ldots, K\}$ with a Dirichlet prior. In particular,

$$\ell(Y_{N(p)} | X_{N(p)}) = \frac{\Gamma(\alpha)}{\Gamma(\frac{\alpha}{K})^K} \frac{\prod_{k=1}^K \Gamma(m_{pk} + \frac{\alpha}{K})}{\Gamma(\sum_{k=1}^K m_{pk} + \alpha)}, \tag{3}$$

where m_{pk} denotes the number of labels $y_n = k$ among those $n \in N(p)$ and α is the concentration parameter of the symmetric Dirichlet prior. Generalisations to other likelihood functions based on conjugate pairs of exponential families are straightforward.

The final piece of the model is the prior density $h(\mathcal{T}|X)$ over decision trees. In order to make straightforward comparisons with existing algorithms, we adopt the model proposed by Chipman et al. (1998). In this model, the prior distribution of the latent tree is defined *conditionally* on the given input vectors X (see Section 5 for a discussion of this dependence on X and its effect on the exchangeability of the labels). Informally, the tree is grown starting at the root, and each new node either splits and grows two children (turning the node into an internal node) or stops (leaving it a leaf) stochastically.

We now describe the generative process more precisely in terms of a Markov chain capturing the construction

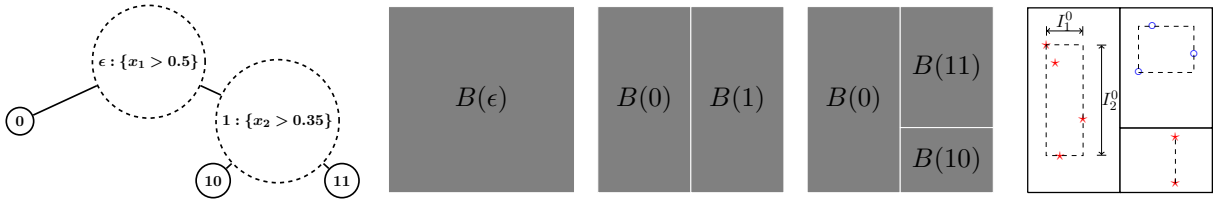

Figure 1. A decision tree $\mathcal{T} = (\mathsf{T}, \kappa, \tau)$ represents a hierarchical partitioning of a space. Here, the space is the unit square and the tree T contains the nodes $\{\epsilon, 0, 1, 10, 11\}$. The root node ϵ represents the whole space $B(\epsilon) = \mathbb{R}^D$, while its two children 0 and 1, represent the two halves of the cut $(\kappa(\epsilon), \tau(\epsilon)) = (1, 0.5)$, where $\kappa(\epsilon) = 1$ represents the dimension of the cut, and $\tau(\epsilon) = 0.5$ represents the location of the cut along that dimension. (The origin is at the bottom left of each figure, and the x-axis is dimension 1. The red stars and blue circles represent observed data points.) The second cut, $(\kappa(1), \tau(1)) = (2, 0.35)$, splits the block $B(1)$ into the two halves $B(11)$ and $B(10)$. When defining the prior over decision trees given by Chipman et al. (1998), it will be necessary to refer to the "extent" of the data in a block. E.g., I_1^0 and I_2^0 are the extent of the data in dimensions 1 and 2, respectively, in block $B(0)$. For each node p, the set D^p contains those dimensions with non-trivial extent. Here, $D^0 = \{1, 2\}$, but $D^{10} = \{2\}$, because there is no variation in dimension 1.

of a decision tree in stages, beginning with the trivial tree $\mathsf{T}_0 = \{\epsilon\}$ containing only the root node. At each stage i, T_i is produced from T_{i-1} by choosing one leaf in T_{i-1} and either *growing* two children nodes or *stopping* the leaf. Once stopped, a leaf is ineligible for future growth. The identity of the chosen leaf is *deterministic*, while the choice to grow or stop is *stochastic*. The process proceeds until all leaves are stopped, and so each node is considered for expansion exactly once throughout the process. This will be seen to give rise to a finite sequence of decision trees $\mathcal{T}_i = (\mathsf{T}_i, \kappa_i, \tau_i)$ once we define the associated cut functions κ_i and τ_i. We will use this Markov chain in Section 3 as scaffolding for a sequential Monte Carlo algorithm. A similar approach was employed by Taddy et al. (2011) in the setting of online Bayesian decision trees. There are similarities also with the *bottom-up* SMC algorithms by Teh et al. (2008) and Bouchard-Côté et al. (2012).

We next describe the rule for stopping or growing nodes, and the distribution of cuts. Let p be the node chosen at some stage of the generative process. If the input vectors $X_{N(p)}$ are all identical, then the node stops and becomes a leaf. (Chipman et al. chose this rule because no choice of cut to the block $B(p)$ would result in both children containing at least one input vector.) Otherwise, let D^p be the set of dimensions along which $X_{N(p)}$ varies, and let $I_d^p = [\min_{n \in N(p)} x_{nd}, \max_{n \in N(p)} x_{nd}]$ be the range of the input vectors along dimension $d \in D^p$. (See last subfigure of Figure 1.) Under the Chipman et al. model, the probability that node p is split is

$$\frac{\alpha_s}{(1 + |p|)^{\beta_s}}, \qquad \alpha_s \in (0, 1), \ \beta_s \in [0, \infty), \tag{4}$$

where $|p|$ is the depth of the node, and α_s and β_s are parameters governing the shape of the resulting tree. For larger α_s and smaller β_s the typical trees are larger, while the deeper p is in the tree the less likely it will be cut. If p is cut, the dimension $\kappa(p)$ and then location $\tau(p)$ of the cut are sampled uniformly from D^p and $I^p_{\kappa(p)}$, respectively. Note that the choice for the support of the distribution over cut dimensions and locations are such that both children of p will, with probability one, contain at least one input vector. Finally, the choices of whether to grow or stop, as well the cut dimensions and locations, are conditionally independent across different subtrees.

To complete the generative model, we define $\mathsf{T} = \mathsf{T}_\eta$, $\kappa = \kappa_\eta$ and $\tau = \tau_\eta$, where η is the first stage such that all nodes are stopped. We note that $\eta < 2N$ with probability one because each cut of a node p produces a non-trivial partition of the data in the block, and a node with one data point will be stopped instead of cut. The conditional density of the decision tree $\mathcal{T} = (\mathsf{T}, \kappa, \tau)$ can now be expressed as

$$h(\mathsf{T}, \kappa, \tau | X) = \prod_{p \in \partial \mathsf{T}} \Big(1 - \frac{\alpha_s}{(1 + |p|)^{\beta_s}}\Big)^{\mathbb{1}(|D^p| > 0)} \times \prod_{p \in \mathsf{T} \setminus \partial \mathsf{T}} \frac{\alpha_s}{(1 + |p|)^{\beta_s}} \frac{1}{|D^p|} \frac{1}{|I^p_{\kappa(p)}|}. \tag{5}$$

Note that the prior distribution of $\mathcal{T}$ does not depend on the deterministic rule for choosing a leaf at each stage. However this choice will have an effect on the bias/variance of the corresponding SMC algorithm.

3. Sequential Monte Carlo (SMC) for Bayesian decision trees

In this section we describe an SMC algorithm for approximating the posterior distribution over the decision tree $(\mathsf{T}, \kappa, \tau)$ given the labeled training data

(X, Y). (We refer the reader to (Cappé et al., 2007) for an excellent overview of SMC techniques.) The approach we will take is to perform particle filtering following the sequential description of the prior. In particular, at stage i, the particles approximate a modified posterior distribution where the prior on $(\mathsf{T}, \kappa, \tau)$ is replaced by the distribution of $(\mathsf{T}_i, \kappa_i, \tau_i)$, i.e., the process truncated at stage i.

Let E_i denote the set of unstopped leaves at stage i, all of which are eligible for expansion. An important freedom we have in our SMC algorithm is the choice of which candidate leaf (or set $C_i \subseteq E_i$ of candidate leaves) to consider expanding. In order to avoid "multipath" issues (Del Moral et al., 2006, §3.5) which lead to high variance, we fix a *deterministic* rule for choosing $C_i \subseteq E_i$. (Multiple candidates are expanded or stopped in turn, independently.) This rule can be a function of (X, Y) and the state of the current particle, as the correctness of resulting approximation is unaffected. We evaluate two choices in experiments: first, the rule $C_i = E_i$ where we consider expanding all eligible nodes; and second, the rule where C_i contains a single node chosen in a breadth-first (i.e., oldest first) manner from E_i.

We may now define the sequence $(\mathbb{P}_i^Y)$ of **target distributions**. Recall the sequential process defined in Section 2. If the generative process for the decision tree has not completed by stage i, the process has generated $(\mathsf{T}_i, \kappa_i, \tau_i)$ along with E_i, capturing which leaves in T_i have been considered for expansion in previous stages already and which have not. Let $\mathcal{T}_i = (\mathsf{T}_i, \kappa_i, \tau_i, E_i)$ be the variables generated on stage i, and write $\mathbb{P}$ for the prior distribution on the sequence $(\mathcal{T}_i)$. We construct the target distribution $\mathbb{P}_i^Y$ as follows: Given $\mathcal{T}_i$, we generate labels Y' with likelihood $g(Y'|\mathcal{T}_i, X)$, i.e., as if $(\mathsf{T}_i, \kappa_i, \tau_i)$ were the complete decision tree. We then define $\mathbb{P}_i^Y$ to be the conditional distribution of $\mathcal{T}_i$ given $Y' = Y$. That is, $\mathbb{P}_i^Y$ is the posterior with a truncated prior.

In order to complete the description of our SMC method, we must define **proposal kernels** $(\mathbb{Q}_i)$ that sample approximations for the ith stage given values for the $(i-1)$th stage. As with our choice of C_i, we have quite a bit of freedom. In particular, the proposals can depend on the training data (X, Y). An obvious choice is to take $\mathbb{Q}_i$ to be the conditional distribution of $\mathcal{T}_i$ given $\mathcal{T}_{i-1}$ under the prior, i.e., setting $\mathbb{Q}_i(\mathcal{T}_i \,|\, \mathcal{T}_{i-1}) = \mathbb{P}(\mathcal{T}_i \,|\, \mathcal{T}_{i-1})$. Informally, this choice would lead us to propose extensions to trees at each stage of the algorithm by sampling from the prior, so we will refer to this as the **prior proposal** kernel (aka the Bayesian bootstrap filter (Gordon et al., 1993)).

We consider two additional proposal kernels: The first,

$$\mathbb{Q}_i(\mathcal{T}_i \,|\, \mathcal{T}_{i-1}) = \mathbb{P}_i^Y(\mathcal{T}_i \,|\, \mathcal{T}_{i-1}), \tag{6}$$

is called the **(one-step) optimal proposal** kernel because it would be the optimal kernel assuming that the ith stage were the final stage. We return to discuss this kernel in Section 3.1. The second alternative, which we will refer to as the **empirical proposal** kernel, is a small modification to the *prior* proposal, differing only in the choice of the split point τ. Recall that, in the prior, $\tau_i(p)$ is chosen uniformly from the interval $I^p_{\kappa_i(p)}$. This ignores the empirical distribution given by the input data $X_{N(p)}$ in the partition. We can account for this by first choosing, uniformly at random, a pair of adjacent data points along feature dimension $\kappa_i(p)$, and then sampling a cut $\tau_i(p)$ uniformly from the interval between these two data points.

The pseudocode for our proposed SMC algorithm is given in Algorithm 1 (Appendix A). Note that the SMC framework only requires us to compute the density of $\mathcal{T}_i$ under the target distribution up to a normalization constant. In fact, the SMC algorithm produces an estimate of the normalization constant, which, at the end of the algorithm, is equal to the marginal probability of the labels Y given X, with the latent decision tree $\mathcal{T}$ marginalized out. In general, the joint density of a Markov chain can be hard to compute, but because the set of nodes C_i considered at each stage is a deterministic function of $\mathcal{T}_i$, the path $(\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_{i-1})$ taken is a deterministic function of $\mathcal{T}_i$. As a result, the joint density is simply a product of probabilities for each stage. The same property holds for the proposal kernels defined above because they use the same candidate set C_i, and have the same support as $\mathbb{P}$. These properties justify the equations in Algorithm 1.

3.1. The one-step optimal proposal kernel

In this section we revisit the definition of the one-step *optimal* proposal kernel. While the *prior* and *empirical* proposal kernels are relatively straightforward, the one-step *optimal* proposal kernel is defined in terms of an additional conditioning on the labels Y, which we now study in greater detail.

Recall that the one-step *optimal* proposal kernel $\mathbb{Q}_i$ is given by $\mathbb{Q}_i(\mathcal{T}_i \,|\, \mathcal{T}_{i-1}) = \mathbb{P}_i^Y(\mathcal{T}_i \,|\, \mathcal{T}_{i-1})$. To begin, we note that, conditionally on $\mathcal{T}_{i-1}$ and Y, the subtrees rooted at each node $p \in C_{i-1}$ are independent. This follows from the fact that the likelihood of Y given $\mathcal{T}_i$ factorizes over the leaves. Thus, the proposal's probability density is

$$\mathbb{Q}_i(\mathcal{T}_i|\mathcal{T}_{i-1}) = \prod_{p \in C_{i-1}} Q_i(\rho_{i,p}, \kappa_i(p), \tau_i(p)), \tag{7}$$

where Q_i is the probability density of the cuts at node p under $\mathbb{Q}_i$, and $\rho_{i,p}$ denotes whether the node was split or not. On the event we split a node $p \in C_{i-1}$, if we condition further on $\kappa_i(p)$ and $\rho_{i,p}$, we note that the conditional likelihood of $Y_{N(p)}$, when viewed as a function of the split $\tau_i(p)$, is piecewise constant, and in particular, only changes when the split crosses an example. It follows that we can sample from this proposal by first considering the discrete choice of an interval, and then sampling uniformly at random from within the interval, as with the *empirical* proposal. Some algebra shows that

$$Q_i(\rho_{i,p} = \text{stop}) \propto \left(1 - \frac{\alpha_s}{(1+|p|)^{\beta_s}}\right) \ell(Y_{N(p)}|X_{N(p)}),$$

$$Q_i(\rho_{i,p} = \text{split}, \kappa_i(p), \tau_i(p)) \propto \frac{\alpha_s}{(1+|p|)^{\beta_s}} \frac{1}{|D^p|} \frac{1}{|I^p_{\kappa_i(p)}|} \times \prod_{j=0,1} \ell(Y_{N(pj)}|X_{N(pj)}).$$

3.2. Computational complexity

Let U_d denote the number of unique values in dimension d, N_p denote the number of training data points at node p and $\eta^{(m)}$ denote the number of nodes in particle m. For all the SMC algorithms, the space complexity is $\mathcal{O}(MN) + \mathcal{O}(\sum_d U_d) + \mathcal{O}(\sum_m \eta^{(m)})$. The time complexity is $\mathcal{O}(DN \log N) + M \sum_p \mathcal{O}(2DN_p + N_p)$ for *prior* and *empirical* proposals and $M \sum_p \big(D\mathcal{O}(N_p \log N_p) + N_p\big)$ for the *optimal* proposal. The *optimal* proposal typically requires higher computational cost per particle, but fewer number of particles than the *prior* and *empirical* proposals.

4. Experiments

In this section, we experimentally evaluate the design choices of the SMC algorithm (proposal, expansion strategy, number of particles and "islands") on real world datasets. In addition, we compare the performance of SMC to the most popular MCMC method for Bayesian decision tree learning (Chipman et al., 1998), as well as CART, a popular (non-Bayesian) tree induction algorithm. We evaluate all the algorithms on the following datasets from the UCI ML repository (Asuncion & Newman, 2007):

- MAGIC gamma telescope data 2004 (*magic-04*): $N = 19020$, $D = 10$, $K = 2$.
- Pen-based recognition of handwritten digits (*pen-digits*): $N = 10992$, $D = 16$, $K = 10$.

Previous work has focused mainly on small datasets (e.g., the Wisconsin breast cancer database used by Chipman et al. (1998) has 683 data points). We chose the above datasets to illustrate the scalability of our approach. For the *pen-digits* dataset, we used the predefined training/test splits, while for the other datasets, we split the datasets randomly into a training set and a test set containing approximately 70% and 30% of the data points respectively.

We implemented our scripts in Python and applied similar software optimization techniques to SMC and MCMC scripts.[1] Our experiments were run on a cluster with machines of similar processing power.

4.1. Design choices in the SMC algorithm

In these set of experiments, we fix the hyperparameters to $\alpha = 5.0, \alpha_s = 0.95, \beta_s = 0.5$ and compare the predictive performance of different configurations of the SMC algorithm for this fixed model. Under the prior, these values of α_s, β_s produce trees whose mean depth and number of nodes are 5.1 and 18.5, respectively. Given M particles, we use an effective sample size (ESS) threshold of $M/10$ and set the maximum number of stages to 5000 (although the algorithms never reached this number).

4.1.1. Proposal choice and node expansion

We consider the SMC algorithm proposed in Section 3 under two proposals: *optimal* and *prior*. (The *empirical* proposal performed similar to the *prior* proposal and hence we do not report those results here.) We consider two strategies for choosing C_i, i.e., the list of nodes considered for expansion at stage i: (i) *node-wise expansion*, where a single node is considered for expansion per stage (i.e., C_i is a singleton chosen deterministically from eligible nodes E_i), and (ii) *layer-wise expansion*, where all nodes at a particular depth are considered for expansion simultaneously (i.e., $C_i = E_i$). For *node-wise* expansion, we evaluate two strategies for selecting the node deterministically from C_i: (i) breadth-first priority, where the oldest node is picked first, and (ii) marginal-likelihood based priority, where we expand the node with the lowest marginal likelihood. Both of these priority schemes performed similarly; hence we report only the results for breadth-first priority. We use multinomial resampling in our experiments. We also evaluated systematic resampling (Douc et al., 2005) but found that the performance was not significantly different.

We report the log predictive probability on test data as a function of runtime and of the number of particles (similar trends are observed for test accuracy;

[1] The scripts can be downloaded from the authors' webpages.

see Appendix B). The times reported do not account for prediction time. We average the numbers over 10 random initializations and report standard deviations. The results are shown in Figure 2. In summary, we observe the following:

Node-wise expansion outperforms *layer-wise* expansion for *prior* proposal. The *prior* proposal does not account for likelihood; one could think of the resampling steps as 'correction steps' for the sub-optimal decisions sampled from the *prior* proposal. Because *node-wise* expansion can potentially resample at every stage, it can correct individual bad decisions immediately, whereas *layer-wise* expansion cannot. In particular, we have observed that *layer-wise* expansion tends to produce shallower trees compared to *node-wise* expansion, leading to poorer performance. This phenomenon can be explained as follows: as the depth of the node increases, the prior probability of stopping increases whereas the posterior probability of stopping might be quite low. In *node-wise* expansion, the resampling step can potentially retain the particles where the node has not been stopped. However, in *layer-wise* expansion, too many nodes might have stopped prematurely and the resampling step cannot 'correct' all these bad decisions easily (i.e., it would require many more particles to sample trees where all the nodes in a layer have not been stopped). Another interesting observation is that *layer-wise* expansion exhibits higher variance: this can be explained by the fact that *layer-wise* expansion samples a greater number of random variables (on average) than *node-wise* before resampling, and so suffers for the same reason that importance sampling can suffer from high variance. Note that both expansion strategies perform similarly for the *optimal* proposal due to the fact that the proposal accounts for the likelihood and resampling does not affect the results significantly. Due to its superior performance, we consider only *node-wise* expansion in the rest of the paper.

The plots on the right side of Figure 2 suggest that the *optimal* proposal requires fewer particles than the *prior* proposal (as expected). However, the per-stage cost of *optimal* proposal is much higher than the *prior*, leading to significant increase in the overall runtime (see Section 3.2 for a related discussion). Hence, the *prior* proposal offers a better predictive performance vs computation time tradeoff than the *optimal* proposal.

The performance of *optimal* proposal saturates very quickly and is near-optimal even when the number of particles is small ($M = 10$).

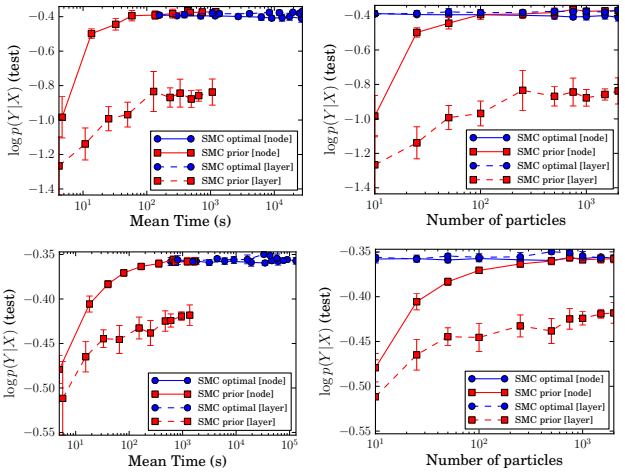

Figure 2. Results on *pen-digits* (top), and *magic-04* (bottom). Left column plots test $\log p(y|x)$ vs runtime, while right column plots test $\log p(y|x)$ vs number of particles. The blue circles and red squares represent *optimal* and *prior* proposals respectively. The solid and dashed lines represent *node-wise* and *layer-wise* proposals respectively.

4.1.2. Effect of irrelevant features

In the next experiment, we test the effect of irrelevant features on the performance of the various proposals. We use the *madelon* dataset[2] for this experiment, in which the data points belong to one of 2 classes and lie in a 500-dimensional space, out of which only 20 dimensions are deemed relevant. The training dataset contains 2000 data points and the test dataset contains 600 data points. We use the validation dataset in the UCI ML repository as our test set because labels are not available for the test dataset.

The setup is identical to the previous section. The results are shown in Figure 3. Here, the *optimal* proposal outperforms the *prior* proposal in both the columns, requiring fewer particles as well as outperforming the *prior* proposal for a given computational budget. While this dataset is atypical (only 4% of the features are relevant), it illustrates a potential vulnerability of the *prior* proposal to irrelevant features.

4.1.3. Effect of the number of islands

Averaging the results of several independent particle filters (aka *islands*) is a way to reduce variance at the cost of bias, compared with running a single, larger filter. In the asymptotic regime, this would not make sense, but as we will see, performance is improved with multiple islands, suggesting we are not yet in the asymptotic regime. In this experiment, we evaluate the effect of the number of islands on the test perfor-

[2] http://archive.ics.uci.edu/ml/datasets/Madelon

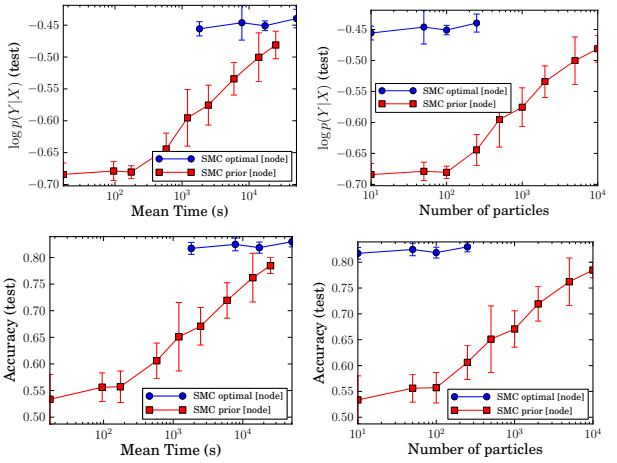

Figure 3. Results on *madelon* dataset: The top and bottom rows display $\log p(y|x)$ and accuracy on the test data against runtime (left) and the number of particles (right) respectively. The blue circles and red squares represent *optimal* and *prior* proposals respectively.

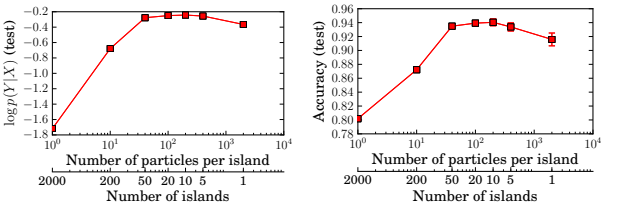

Figure 4. Results on *pen-digits*: Test $\log p(y|x)$ (left) and accuracy (right) vs I and M/I for fixed $M = 2000$.

mance of the *prior* proposal. We fix the total number of particles to 2000 and vary I, the number of islands (and hence, the number of particles per island). Note that all the islands operate on the entire dataset unlike *bagging*. Here, we present results only on the *pen-digits* dataset (see Appendix C for results on the *magic-04* dataset). The results are shown in Figure 4. We observe that (i) the test performance drops sharply if we use fewer than 100 particles per island and (ii) when $M/I \geq 100$, the choices of $I \in [5, 100]$ outperform $I = 1$. Since the islands are independent, the computation across islands is 'embarrassingly parallelizable'.

4.2. SMC vs MCMC

In this experiment, we compare the SMC algorithms to the MCMC algorithm proposed by Chipman et al. (1998), which employs four types of Metropolis-Hastings proposals: *grow* (split a leaf node into child nodes), *prune* (prune a pair of leaf nodes belonging to the same parent), *change* (change the decision rule at a node) and *swap* (swap the decision rule of a parent with the decision rule of the child). In our experiments, we average the MCMC predictions over the trees from all previous iterations.

The experimental setup is identical to Section 4.1, except that we fix the number of islands, $I = 5$. We vary the number of particles for SMC[3] and the number of iterations for MCMC and plot the log predictive probability and accuracy on the test data as a function of runtime. In Figure 5, we observe that SMC (*prior*, *node-wise*) is roughly two orders of magnitude faster than MCMC while achieving similar predictive performance on *pen-digits* and *magic-04* datasets. Although the exact speedup factor depends on the dataset in general, we have observed that **SMC (*prior*, *node-wise*) is at least an order of magnitude faster than MCMC**. The SMC runtimes in Figure 5 are recorded by running the I islands in a serial fashion. As discussed in Section 4.1.3, one could parallelize the computation leading to an additional speedup by a factor of I. In the *pen-digits* dataset, the performance of *prior* proposal seems to drop as we increase M beyond 2000. However, the marginal likelihood on the training data increases with M (see Appendix D). We believe that the deteriorating performance is due to model misspecification (axis-aligned decision trees are hardly the 'right' model for handwritten digits) rather than the inference algorithm itself: 'better' Bayesian inference in a misspecified model might lead to a poorer solution (see (Minka, 2000) for a related discussion).

To evaluate the sensitivity of the trends above to the hyper parameters $\alpha, \alpha_s, \beta_s$, we systematically varied the values of these hyper parameters and repeated the experiment. The results are qualitatively similar. See Appendix E for additional information.

4.3. SMC vs other existing approaches

The goal of these experiments was to verify that our SMC approximation performed as well as the "gold standard" MCMC algorithms most commonly used in the Bayesian decision tree learning setting. Indeed, our results suggest that, for a fraction of the computational budget, we can achieve a comparable level of accuracy. In this final experiment, we re-affirm that the Bayesian algorithms are competitive in accuracy with the classic CART algorithm. (There are many other comparisons that one could pursue and other authors have already performed such comparisons. E.g., Taddy et al. (2011) demonstrated that their tree structured models yield similar performance as Gaussian

[3]We fix $I = 5$ so that the minimum value of M ($= 100$) corresponds to $M/I = 20$ particles per island. Further improvements could be obtained by 'adapting' I to M as discussed in Section 4.1.3.

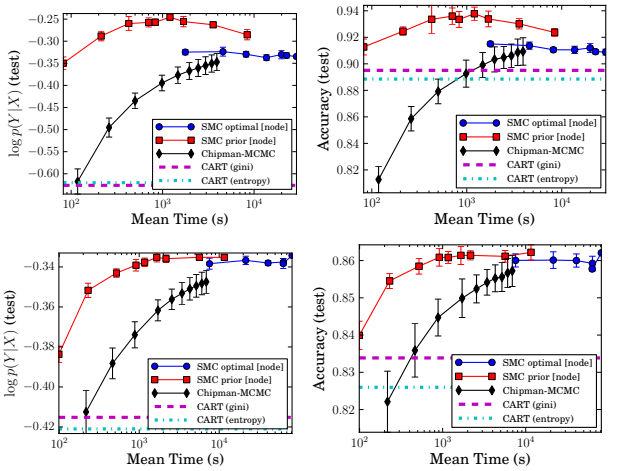

Figure 5. Results on *pen-digits* (top row), and *magic-04* (bottom row). Left column plots test $\log p(y|x)$ vs runtime, while right column plots test accuracy vs runtime. The blue cirlces, red squares and black diamonds represent *optimal*, *prior* proposals and MCMC respectively.

processes and random forests.) We used the CART implementation provided by *scikit-learn* (Pedregosa et al., 2011) with two criteria: *gini purity* and *information gain* and set `min_samples_leaf` $= 10$ (minimum number of data points at a leaf node).[4] In addition, we performed Laplacian smoothing on the probability estimates from CART using the same α as for the Bayesian methods. Our Python implementation of SMC takes about 50-100x longer to achieve the same test accuracy as the highly-optimized implementation of CART. For this reason, we plot CART accuracy as a horizontal bar. The accuracy and log predictive probability on test data are shown in Figure 5. The Bayesian decision tree frameworks achieve similar (or better) test accuracy to CART, and outperform CART significantly in terms of the predictive likelihood. SMC delivers the benefits of having an approximation to the posterior, but in a fraction of the time required by existing MCMC methods.

5. Discussion and Future work

We have proposed a novel class of Bayesian inference algorithms for decision trees, based on the sequential Monte Carlo framework. The algorithms mimic classic top-down algorithms for learning decision trees, but use "local" likelihoods along with resampling steps to guide tree growth. We have shown good computational and statistical performances, especially compared with a state-of-the-art MCMC inference algorithm. Our algorithms are easier to implement than their MCMC counterparts, whose efficient implementations require sophisticated book-keeping.

We have also explored various design choices leading to different SMC algorithms. We have found that expanding too many nodes simultaneously degraded performance, and more sophisticated ways of choosing nodes surprisingly did not improve performance. Finally, while the one-step *optimal* proposal often required fewer particles to achieve a given accuracy, it was significantly more computationally intensive than the *prior* proposal, leading to a less efficient algorithm overall on datasets with few irrelevant input dimensions. As the number of irrelevant dimensions increased the balance tipped in favour of the *optimal* proposal. An interesting direction of exploration is to devise some way to interpolate between the *prior* and *optimal* proposals, getting the best of both worlds.

The model underlying this work assumes that the data is explained by a single tree. In contrast, many uses of decision trees, e.g., random forests, bagging, etc., can be interpreted as working within a model class where the data is explained by a collection of trees. Bayesian additive regression trees (BART) (Chipman et al., 2010) are such a model class. Prior work has considered MCMC techniques for posterior inference (Chipman et al., 2010). A significant but important extension of this work would be to tackle additive combinations of trees, potentially in a way that continues to mimic classic algorithms.

Finally, in order to more closely match existing work in Bayesian decision trees, we have used a prior over decision trees that depends on the input data X. This has the undesirable side-effect of breaking exchangeability in the model, making it incoherent with respect to changing dataset sizes and to working with online data streams. One solution is to use an alternative prior for decision trees, e.g., based on the Mondrian process (Roy & Teh, 2009), whose projectivity would re-establish exchangeability while allowing for efficient posterior *computations* that depend on data.

Acknowledgments

We would like to thank Charles Blundell, Arnaud Doucet, David Duvenaud, Jan Gasthaus, Hong Ge, Zoubin Ghahramani, and James Robert Lloyd for helpful discussions and feedback on drafts. DMR is supported by a Newton International Fellowship and Emmanuel College. BL and YWT gratefully acknowledge generous funding from the Gatsby Charitable Foundation.

[4] Lower values (`min_samples_leaf` $= 1, 5$) tend to yield slightly higher test accuracies (comparable to SMC and MCMC) but much lower predictive probabilities.

References

Anagnostopoulos, C. and Gramacy, R.B. Dynamic trees for streaming and massive data contexts. *arXiv preprint arXiv:1201.5568*, 2012.

Asuncion, A. and Newman, D. J. UCI machine learning repository. http://www.ics.uci.edu/~mlearn/MLRepository.html, 2007.

Bouchard-Côté, A., Sankararaman, S., and Jordan, M. I. Phylogenetic inference via sequential monte carlo. *Systematic biology*, 61(4):579–593, 2012.

Breiman, L. Random forests. *Machine Learning*, 45: 5–32, 2001.

Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. *Classification and regression trees*. Chapman & Hall/CRC, 1984.

Buntine, W. Learning classification trees. *Stat. Comput.*, 2:63–73, 1992.

Cappé, O., Godsill, S. J., and Moulines, E. An overview of existing methods and recent advances in sequential Monte Carlo. *Proc. IEEE*, 95(5):899–924, 2007.

Chipman, H. and McCulloch, R. E. Hierarchical priors for Bayesian CART shrinkage. *Stat. Comput.*, 10(1): 17–24, 2000.

Chipman, H. A., George, E. I., and McCulloch, R. E. Bayesian CART model search. *J. Am. Stat. Assoc.*, pp. 935–948, 1998.

Chipman, H. A., George, E. I., and McCulloch, R. E. BART: Bayesian additive regression trees. *Ann. Appl. Stat.*, 4(1):266–298, 2010.

Del Moral, P., Doucet, A., and Jasra, A. Sequential Monte Carlo samplers. *J. R. Stat. Soc. Ser. B Stat. Methodol.*, 68(3):411–436, 2006.

Denison, D. G. T., Mallick, B. K., and Smith, A. F. M. A Bayesian CART algorithm. *Biometrika*, 85(2): 363–377, 1998.

Douc, R., Cappé, O., and Moulines, E. Comparison of resampling schemes for particle filtering. In *Image Sig. Proc. Anal.*, pp. 64–69, 2005.

Friedman, J. H. Greedy function approximation: a gradient boosting machine. *Ann. Statist*, 29(5): 1189–1232, 2001.

Gordon, N. J., Salmond, D. J., and Smith, A. F. M. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *Radar Sig. Proc., IEE Proc. F*, 140(2):107–113, 1993.

Minka, T. P. Bayesian model averaging is not model combination. MIT Media Lab note. http://research.microsoft.com/en-us/um/people/minka/papers/bma.html, 2000.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and E., Duchesnay. Scikit-learn: Machine Learning in Python. *J. Machine Learning Res.*, 12: 2825–2830, 2011.

Quinlan, J. R. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

Quinlan, J. R. *C4.5: programs for machine learning*. Morgan Kaufmann, 1993.

Roy, D. M. and Teh, Y. W. The Mondrian process. In *Adv. Neural Information Proc. Systems*, volume 21, pp. 1377–1384, 2009.

Taddy, M. A., Gramacy, R. B., and Polson, N. G. Dynamic trees for learning and design. *J. Am. Stat. Assoc.*, 106(493):109–123, 2011.

Teh, Y. W., Daumé III, H., and Roy, D. M. Bayesian agglomerative clustering with coalescents. In *Adv. Neural Information Proc. Systems*, volume 20, 2008.

Wu, Y., Tjelmeland, H., and West, M. Bayesian CART: Prior specification and posterior simulation. *J. Comput. Graph. Stat.*, 16(1):44–66, 2007.

Supplementary material

A. SMC algorithm

Algorithm 1 SMC for Bayesian decision tree learning

Inputs: Training data (X, Y)
Number of particles M
Initialize: $\mathsf{T}_0^{(m)} = E_0^{(m)} = \{\epsilon\}$
$\tau_0^{(m)} = \kappa_0^{(m)} = \emptyset$
$w_0^{(m)} = f(Y|\mathcal{T}_0^{(m)})$
$W_0 = \sum_m w_0^{(m)}$

for $i = 1 : \text{MAX-STAGES}$ **do**
  for $m = 1 : M$ **do**
    Sample $\mathcal{T}_i^{(m)}$ from $\mathbb{Q}_i(\cdot \mid \mathcal{T}_{i-1}^{(m)})$
      where $\mathcal{T}_i^{(m)} := (\mathsf{T}_i^{(m)}, \kappa_i^{(m)}, \tau_i^{(m)}, E_i^{(m)})$
    Update weights: (Here $\mathbb{P}, \mathbb{Q}_i$ denote their densities.)

$$w_i^{(m)} = \frac{\mathbb{P}(\mathcal{T}_i^{(m)})\, g(Y \mid \mathcal{T}_i^{(m)}, X)}{\mathbb{Q}_i(\mathcal{T}_i^{(m)} \mid \mathcal{T}_{i-1}^{(m)})\, \mathbb{P}(\mathcal{T}_{i-1}^{(m)})} \tag{8}$$

$$= w_{i-1}^{(m)} \frac{\mathbb{P}(\mathcal{T}_i^{(m)} \mid \mathcal{T}_{i-1}^{(m)})}{\mathbb{Q}_i(\mathcal{T}_i^{(m)} \mid \mathcal{T}_{i-1}^{(m)})} \frac{g(Y \mid \mathcal{T}_i^{(m)}, X)}{g(Y \mid \mathcal{T}_{i-1}^{(m)}, X)} \tag{9}$$

  end for
  Compute normalization: $W_i = \sum_m w_i^{(m)}$
  Normalize weights: $(\forall m)\, \bar{w}_i^{(m)} = w_i^{(m)} / W_i$
  if $\left(\sum_m (\bar{w}_i^{(m)})^2\right)^{-1} <$ ESS-THRESHOLD **then**
    $(\forall m)$ Resample indices j_m from $\sum_{m'} \bar{w}_i^{(m')} \delta_{m'}$
    $(\forall m)\ \mathcal{T}_i^{(m)} \leftarrow \mathcal{T}_i^{(j_m)}$; $w_i^{(m)} \leftarrow W_i / M$
  end if
  if $(\forall m)\, E_i^{(m)} = \emptyset$ **then**
    exit for loop
  end if
end for
return Estimated marginal probability W_i/M and weighted samples $\{w_i^{(m)}, \mathsf{T}_i^{(m)}, \kappa_i^{(m)}, \tau_i^{(m)}\}_{m=1}^M$.

B. Effect of SMC proposal and expansion strategy on test accuracy

The results are shown in Figure 6.

C. Effect of the number of islands: *magic-04* dataset

The results are shown in Figure 7.

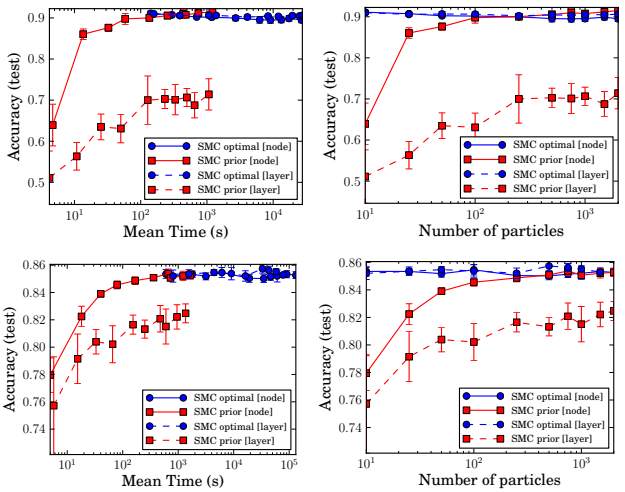

Figure 6. Results on *pen-digits* (top), and *magic-04* (bottom). Left column plots test accuracy vs runtime, while right column plots test accuracy vs number of particles. The blue circles and red squares represent *optimal* and *prior* proposals respectively. The solid and dashed lines represent *node-wise* and *layer-wise* proposals respectively.

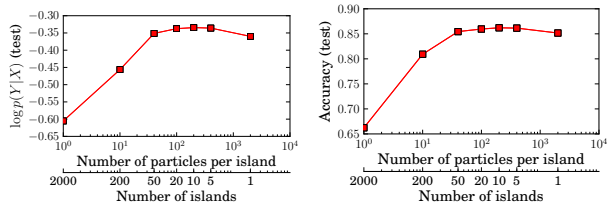

Figure 7. Results on *magic-04*: Test $\log p(y|x)$ (left) and accuracy (right) vs I and M/I for fixed $M = 2000$.

D. Marginal likelihood

The log marginal likelihood of the training data for different proposals is shown in Figure 8. As the number of particles increases, the log marginal likelihood of *prior* and *optimal* proposals converge to the same value (as expected).

E. Sensitivity of results to choice of hyperparameters

In this experiment, we evaluate the sensitivity of the runtime vs predictive performance comparison between SMC (*prior* and *optimal* proposals), MCMC and CART to the choice of hyper parameters α (Dirichlet concentration parameter) and α_s, β_s (tree priors). We consider only *node-wise* expansion since it consistently outperformed *layer-wise* expansion in our previous experiments. In the first variant, we fix $\alpha = 5.0$ (since we do not expect it to affect the

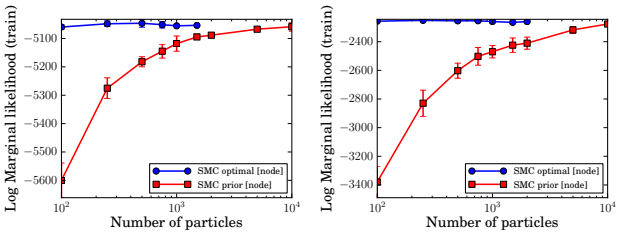

Figure 8. Results on *pen-digits* (left), and *magic-04* (right). Mean log marginal likelihood (i.e., mean $\log p(Y|X)$ for training data averaged across 10 runs) vs number of particles. The blue circles and red squares represent *optimal* and *prior* proposals respectively.

timing results) and vary the hyper parameters from $\alpha_s = 0.95, \beta_s = 0.5$ to $\boldsymbol{\alpha_s = 0.8, \beta_s = 0.2}$ (bold reflects changes) and also consider intermediate configurations $\alpha_s = 0.95, \boldsymbol{\beta_s = 0.2}$ and $\boldsymbol{\alpha_s = 0.8}, \beta_s = 0.5$. In the second variant, we fix $\alpha_s = 0.95, \beta_s = 0.5$ and set $\boldsymbol{\alpha = 1.0}$. Figures 9, 10, 11 and 12 display the results on *pen-digits* (top row), and *magic-04* (bottom row). The left column plots test $\log p(y|x)$ vs runtime, while the right column plots test accuracy vs runtime. The blue circles and red squares represent *optimal* and *prior* proposals respectively. Comparing the results to Figure 5 (in main text), we observe that the trends are qualitatively similar to those observed for $\alpha = 5.0, \alpha_s = 0.95, \beta_s = 0.5$ in Section 4.2 (in main text): (i) SMC consistently offers a better runtime vs predictive performance tradeoff than MCMC, (ii) the *prior* proposal offers a better runtime vs predictive performance tradeoff than the *optimal* proposal, (iii) $\alpha = 1.0$ leads to similar test accuracies as $\alpha = 5.0$ (the predictive probabilities are obviously not comparable).

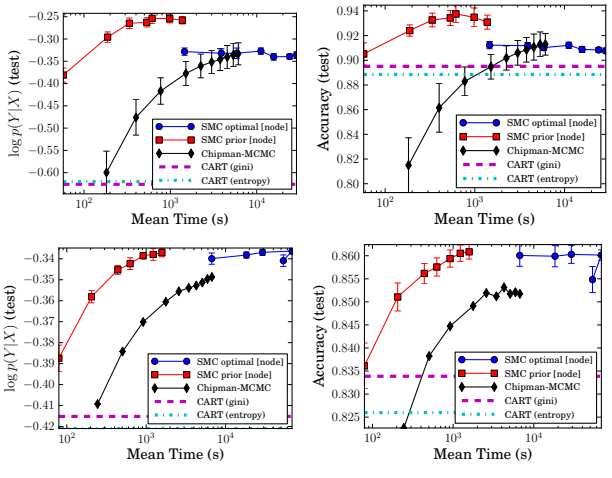

Figure 9. Hyperparameters: $\alpha = 5.0, \boldsymbol{\alpha_s = 0.8}, \beta_s = 0.5$ (see main text for additional information).

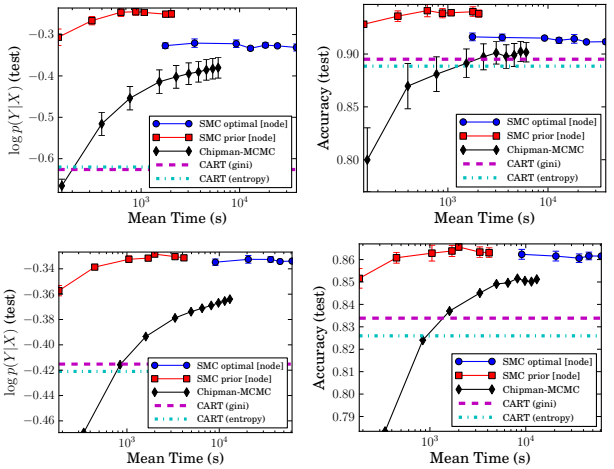

Figure 10. Hyperparameters: $\alpha = 5.0, \alpha_s = 0.95, \boldsymbol{\beta_s = 0.2}$ (see main text for additional information).

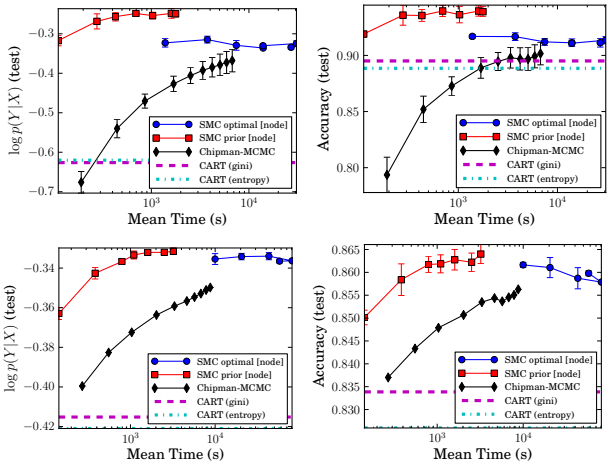

Figure 11. Hyperparameters: $\alpha = 5.0, \boldsymbol{\alpha_s = 0.8, \beta_s = 0.2}$ (see main text for additional information).

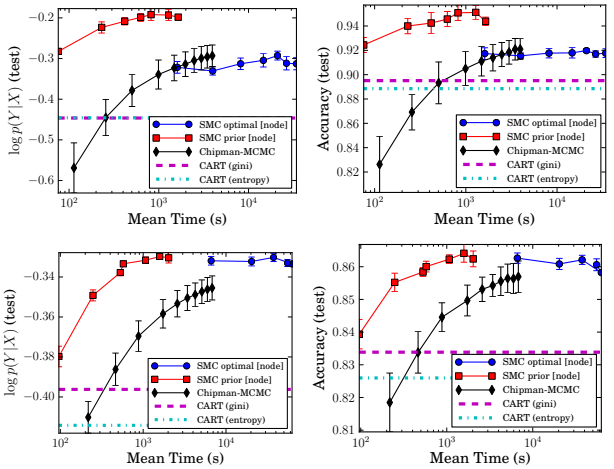

Figure 12. Hyperparameters: $\boldsymbol{\alpha = 1.0}, \alpha_s = 0.95, \beta_s = 0.5$ (see main text for additional information).