Decision Trees and Forests: A Probabilistic Perspective

Balaji Lakshminarayanan

Gatsby Computational Neuroscience Unit
University College London
Sainsbury Wellcome Centre, 25 Howland St,
London, United Kingdom

THESIS

Submitted for the degree of
Doctor of Philosophy, University College London

2016

I, Balaji Lakshminarayanan, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Abstract

Decision trees and ensembles of decision trees are very popular in machine learning and often achieve state-of-the-art performance on black-box prediction tasks. However, popular variants such as C4.5, CART, boosted trees and random forests lack a probabilistic interpretation since they usually just specify an algorithm for training a model. We take a probabilistic approach where we cast the decision tree structures and the parameters associated with the nodes of a decision tree as a probabilistic model; given labeled examples, we can train the probabilistic model using a variety of approaches (Bayesian learning, maximum likelihood, etc). The probabilistic approach allows us to encode prior assumptions about tree structures and share statistical strength between node parameters; furthermore, it offers a principled mechanism to obtain probabilistic predictions which is crucial for applications where uncertainty quantification is important.

Existing work on Bayesian decision trees relies on Markov chain Monte Carlo which can be computationally slow and suffer from poor mixing. We propose a novel sequential Monte Carlo algorithm that computes a particle approximation to the posterior over trees in a top-down fashion. We also propose a novel sampler for Bayesian additive regression trees by combining the above top-down particle filtering algorithm with the Particle Gibbs (Andrieu et al., 2010) framework.

Finally, we propose Mondrian forests (MFs), a computationally efficient hybrid solution that is competitive with non-probabilistic counterparts in terms of speed and accuracy, but additionally produces well-calibrated uncertainty estimates. MFs use the Mondrian process (Roy and Teh, 2009) as the randomization mechanism and hierarchically smooth the node parameters within each tree (using a hierarchical probabilistic model and approximate Bayesian updates), but combine the trees in a non-Bayesian fashion. MFs can be grown in an incremental/online fashion and remarkably, the distribution of online MFs is the same as that of batch MFs.

Acknowledgments

I consider myself very fortunate to have been supervised by Yee Whye Teh. He is not only a brilliant scientist, but also an amazing mentor. He encouraged me to pursue different research directions and opened up many opportunities for fruitful collaborations. The running theme of this thesis is the exploration (and exploitation ☺) of connections between computationally efficient tricks in decision tree land and neat mathematical ideas in (non-parametric) Bayesian land. YeeWhye's earlier work (on coalescents, hierarchical Pitman-Yor process and the Mondrian process particularly) served as a great inspiration for the work presented in this thesis, and his insights added the magic touch to Mondrian forests. I learned a lot by working with him, for which I am most grateful.

I would like to thank all my collaborators during my PhD. In particular, Dan Roy has been a close collaborator for the research presented in this thesis and deserves special mention. Dan is also probably part responsible for my Mondrian obsession ☺.

The Gatsby unit is an amazing environment for research and provided me the opportunity to interact with lots of friendly and brilliant people on a daily basis. The talks, reading groups and discussions helped me 'connect the dots' and think critically about my own research. I would like to thank the faculty Arthur Gretton, Maneesh Sahani, Peter Latham and in particular, Peter Dayan, for his inspiring leadership. I would like to thank all the postdocs and fellow students who make Gatsby a special place; I won't name everyone for I might miss someone inadvertently, but I would particularly like to thank Arthur, Bharath, Charles, Dino, Heiko, Jan, Laurence, Loic, Maria, Srini, Wittawat, and Zoltan. I would like to thank Reign and Barry for their help with countless administrative issues, and Faz and John, for their technical support. I would like to thank the Gatsby charitable foundation for funding my studies.

I would like to thank my family for their love and support. I would like to thank my friends Sai, Krishna, Karthik and Bharath for all the fun ☺. Finally and most importantly, I would like to thank my wife Anusha for her love and patience. Without her, this thesis would not have been possible.

Contents

List of figures

List of tables

List of algorithms

Chapter 1

Outline

Decision trees are a very popular tool in machine learning and statistics for prediction tasks (e.g. classification and regression). In a nutshell, learning a decision tree from training data involves two steps: (i) learning an hierarchical, tree-structured partitioning of the input space and (ii) learning to predict the label within each leaf node. During prediction stage, we simply traverse down the decision tree from the root to the leaf node and predict the label. Popular decision tree induction algorithms such as CART (Breiman et al., 1984) and C4.5 (Quinlan, 1993) have been named amongst the top 10 algorithms in data mining (Wu et al., 2008). The main advantage of decision trees is that they are computationally fast to train and test. Another advantage of decision trees is that they are well-suited for datasets with mixed attribute types (e.g. binary, categorical, real-valued attributes). Moreover, they deliver good accuracy and are interpretable (at least on simple problems), hence they are very popular in practical applications.

While decision trees are powerful, they are prone to over-fitting and require heuristics to limit their complexity (e.g. limiting the maximum depth or pruning the learned decision tree on a validation data set) in order to minimize their generalization error. A useful way to think about the over fitting issue is in terms of bias variance tradeoff, using the tree depth as a complexity measure (as deeper trees can capture more complex interactions). Deep decision trees exhibit low bias as they can potentially memorize the training dataset, however they exhibit high variance, i.e. a decision tree algorithm trained on two different training datasets (from the same 'population' distribution) would produce very different decision trees; hence, decision trees are also referred to as *unstable learners*. Another disadvantage of decision trees is that they typically do not produce probabilistic predictions. In many applications (e.g. clinical decision making), it is useful to have a predictor that can quantify predictive uncertainty instead of just producing a point estimate. The probabilistic approach (Ghahramani, 2015; Murphy, 2012) provides an elegant solution to both of these problems.

Specifically, the Bayesian approach (Bayes and Price, 1763) provides a principled mechanism to prevent over-fitting. The Bayesian approach is conceptually very simple.

First, we introduce a *prior* over decision trees (e.g. a prior that prefers shallow trees) and the leaf node parameters (i.e. the parameters that predict the label within each leaf node). Next, we define a *likelihood* which measures how well a decision tree explains the given training data. Finally, we compute the Bayesian posterior over decision trees and the node parameters. During prediction, the predictions of trees are weighted according to their weights according to the posterior distribution. This process is known as *Bayesian model averaging* (Hoeting et al., 1999) and accounts for the uncertainty in the model (the model is the decision tree in this case) instead of picking just one decision tree. Moreover, the Bayesian approach allows us to better quantify predictive uncertainty, by translating model uncertainty into predictive uncertainty. The main disadvantage of the Bayesian approach is the computational complexity. While computing the Bayesian posterior over node parameters is fairly straightforward, computing the exact posterior distribution over trees is infeasible for non-trivial problems and in practice, we have to resort to approximations. Some early examples of Bayesian decision trees are Buntine (1992); Chipman et al. (1998); Denison et al. (1998).

Ensemble learning (Dietterich, 2000), where we combine many predictors / learners, is another way to address over-fitting. Two popular ensemble strategies are *boosting* (Schapire, 1990; Freund et al., 1999) and bootstrap aggregation, more commonly referred to *bagging* (Breiman, 1996) . While ensemble learning can be combined with any learning algorithm, ensembles of decision trees are very popular since decision trees are unstable learners and are computationally fast to train and test. Ensembles of decision trees often achieve state-of-the-art performance in many supervised learning problems (Caruana and Niculescu-Mizil, 2006; Fernández-Delgado et al., 2014). While the combination of boosting and decision trees has been studied by many researchers (cf. (Freund et al., 1999)), the most popular variant in practice is the *gradient boosted decision trees (GBRT)* algorithm proposed by Friedman (2001). While GBRTs are popular in practice, they can over-fit and moreover, they do not produce probabilistic predictions. Chipman et al. (2010) proposed *Bayesian additive regression trees (BART)*, a Bayesian version of boosted decision trees. In his seminal paper, Breiman (2001) proposed *random forests* (RF) which consist of multiple *randomized* decision trees. Some popular strategies for randomizing the individual trees in a random forest are (i) training individual trees on bootstrapped versions of the original dataset, (ii) randomly sampling a subset of the original features before optimizing for split dimension and split location and (iii) randomly sampling candidate pairs of split dimensions and split locations and restricting the search to just these pairs. While random forests were originally proposed for supervised learning, the random forest framework is very flexible and can be extended to other problems such as density estimation, manifold learning and semi-supervised learning (Criminisi et al., 2012). Random forests are less prone to over-fitting, however they do not produce probabilistic predictions. Another disadvantage of random forests is that they are difficult to train incrementally.

In this thesis, we take a probabilistic approach where we cast the decision tree structures

and the parameters associated with the nodes of a decision tree as a probabilistic model. The probabilistic approach allows us to encode prior assumptions about tree structures and share statistical strength between node parameters. Moreover, the probabilistic approach offers a principled mechanism to obtain probabilistic predictions and quantify predictive uncertainty. *The probabilistic view enables us to think about the different sources of uncertainty and understand the computational vs performance trade-offs involved in designing an ensemble of decision trees with desirable properties (high accuracy, fast predictions, probabilistic predictions, efficient online training, etc).*

We make several contributions in this thesis:

- In Chapter 2, we review decision trees and set up the notation. We briefly review ensembles of decision trees, clarify what it means to be Bayesian in this context, and discuss the relative merits of Bayesian and non-Bayesian approaches.
- In Chapter 3, we first present a novel sequential interpretation of the decision tree prior and then propose a top-down particle filtering algorithm for Bayesian learning of decision trees as an alternative to Markov chain Monte Carlo (MCMC) methods. This chapter is based on (Lakshminarayanan et al., 2013), published in ICML 2013, and is joint work with Daniel M. Roy and Yee Whye Teh.
- In Chapter 4, we combine the above top-down particle filtering algorithm with the Particle MCMC framework (Andrieu et al., 2010) and propose PG-BART, a Particle Gibbs sampler for BART. This chapter is based on (Lakshminarayanan et al., 2015), published in AISTATS 2015, and is joint work with Daniel M. Roy and Yee Whye Teh.
- In Chapter 5, we propose a novel random forest called *Mondrian forest (MF)* that leverages tools from the nonparametric-Bayesian literature such as the Mondrian process (Roy and Teh, 2009) and the hierarchical Pitman-Yor process (Teh, 2006). Unlike existing random forests, Mondrian forests produce principled uncertainty estimates, and can be trained online efficiently. This chapter is based on (Lakshminarayanan et al., 2014), published in NIPS 2014, and is joint work with Daniel M. Roy and Yee Whye Teh.
- In Chapter 6, we extend Mondrian forests to regression and demonstrate that MFs outperform approximate Gaussian processes on large-scale regression, and produce better uncertainty estimates than popular decision forests. This chapter is based on (Lakshminarayanan et al., 2016), published in AISTATS 2016, and is joint work with Daniel M. Roy and Yee Whye Teh.
- We conclude in Chapter 7 and discuss avenues for future work.

Chapter 2

Review of decision trees and ensembles of trees

2.1 Problem setup

Given N labeled examples $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N) \in \mathcal{X} \times \mathcal{Y}$ as training data, the task in supervised learning is to predict labels $y \in \mathcal{Y}$ for unlabeled test points $\boldsymbol{x} \in \mathcal{X}$. Since we are interested in probabilistic predictions, our goal is to not just predict a label $y \in \mathcal{Y}$, but to output the distribution $p(y|\boldsymbol{x})$, i.e. the conditional distribution of the label y given the features $\boldsymbol{x}$. For simplicity, we assume that $\mathcal{X} := \mathbb{R}^D$, where D denotes the dimensionality (i.e. the number of features), and restrict our attention to two popular supervised learning scenarios:

- *multi-class classification* (of which *binary classification* is a special case) where $\mathcal{Y} := \{1, \ldots, K\}$ (K denotes the number of classes in this case), and
- *regression* where $\mathcal{Y} := \mathbb{R}$.

Let $\boldsymbol{X}_{1:n} := (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, $Y_{1:n} := (y_1, \ldots, y_n)$, and $\mathcal{D}_{1:n} := (\boldsymbol{X}_{1:n}, Y_{1:n})$. For every subset $A \subseteq \{1, \ldots, N\}$, let $Y_A := \{y_n : n \in A\}$ and similarly for $\boldsymbol{X}_A$ and $\mathcal{D}_A$.

2.2 Decision trees

For our purposes, a **decision tree** on $\mathcal{X}$ will be a hierarchical, axis-aligned, binary partitioning of $\mathcal{X}$ and a rule for predicting the label of test points given training data. The structure of the decision tree is a finite, rooted, strictly binary tree T, i.e., a finite set of **nodes** such that 1) every node j has exactly one **parent** node, except for a distinguished **root** node ϵ which has no parent, and 2) every node j is the parent of exactly zero or two **children** nodes, called the left child $\mathsf{left}(j)$ and the right child $\mathsf{right}(j)$. Denote the leaves of T (those nodes without children) by $\mathsf{leaves}(\mathsf{T})$. Each node

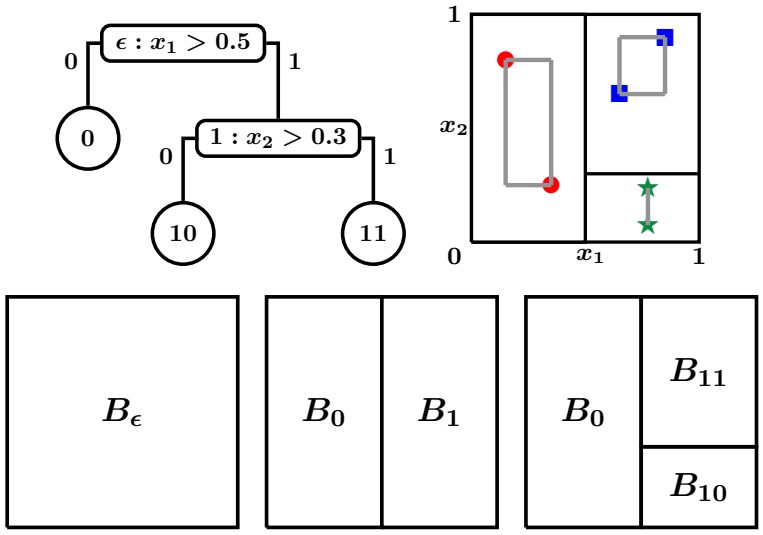


Figure 2.1: A decision tree $\mathcal{T} = (\mathsf{T}, \boldsymbol{\delta}, \boldsymbol{\xi})$ represents a hierarchical partitioning of a space. Here, the space is the unit square and the tree T contains the nodes $\{\epsilon, 0, 1, 10, 11\}$. The root node ϵ represents the whole space $B_\epsilon = \mathbb{R}^D$, while its two children 0 and 1, represent the two halves of the cut $(\delta_\epsilon, \xi_\epsilon) = (1, 0.5)$, where $\delta_\epsilon = 1$ represents the dimension of the cut, and $\xi_\epsilon = 0.5$ 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 circles, green stars and blue squares represent observed data points.) The second cut, $(\delta_1, \xi_1) = (2, 0.35)$, 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. Here, $B_j^x = e_{j1}^x \times e_{j2}^x$ denotes the bounding box of data (shown in gray) in block B_j, where e_{j1}^x and e_{j2}^x are the *e*xtent of the data in dimensions 1 and 2, respectively. For each node j, the set $\mathcal{V}_j$ contains those dimensions with non-trivial extent. Here, $\mathcal{V}_0 = \{1, 2\}$, but $\mathcal{V}_{10} = \{2\}$, because there is no variation in dimension 1.

of the tree $j \in \mathsf{T}$ is associated with a block $B_j \subset \mathbb{R}^D$ of the input space as follows: At the root, we have $B_\epsilon = \mathbb{R}^D$, while each **internal** node $j \in \mathsf{T} \setminus \mathsf{leaves}(\mathsf{T})$ with two children represents a **split** of its parent's block into two halves, with $\delta_j \in \{1, \ldots, D\}$ denoting the dimension of the (axis-aligned) split, and ξ_j denoting the location of the split. In particular,

$$B_{\mathsf{left}(j)} := \{\boldsymbol{x} \in B_j : x_{\delta_j} \leq \xi_j\} \quad \text{and} \quad B_{\mathsf{right}(j)} := \{\boldsymbol{x} \in B_j : x_{\delta_j} > \xi_j\}.$$

We call the tuple $\mathcal{T} = (\mathsf{T}, \boldsymbol{\delta}, \boldsymbol{\xi})$ a **decision tree**. (See Figure 2.1 for more intuition on the representation and notation of decision trees.) Note that the blocks associated with the leaves of the tree form a partition of $\mathbb{R}^D$. We may write $B_j = \big(\ell_{j1}, u_{j1}\big] \times \ldots \times \big(\ell_{jD}, u_{jD}\big]$, where ℓ_{jd} and u_{jd} denote the ℓower and $u$pper bounds, respectively, of the rectangular block B_j along dimension d. Let $\boldsymbol{\ell}_j = \{\ell_{j1}, \ell_{j2}, \ldots, \ell_{jD}\}$ and $\mathbf{u}_j = \{u_{j1}, u_{j2}, \ldots, u_{jD}\}$.

It will be useful to introduce some additional notation. Let $\mathsf{parent}(j)$ denote the parent of node j. Let N_j denote the indices of training data points at node j, i.e., $N_j = \{n \in \{1, \ldots, N\} : \boldsymbol{x}_n \in B_j\}$. Note that both B_j and N_j depend on $\mathcal{T}$, although we have chosen to elide this dependence for notational simplicity. Let $\mathcal{D}_{N_j} = \{\boldsymbol{X}_{N_j}, Y_{N_j}\}$ denote the features and labels of training data points at node j. Let ℓ_{jd}^x and u_{jd}^x denote

the ℓower and $u$pper bounds of training data points (hence the superscript x) respectively in node j along dimension d. Additionally, let $e^x_{jd} = (\ell^x_{jd}, u^x_{jd}]$ denote the $e$xtent of the training data in node j along dimension d. Let $B^x_j = \left(\ell^x_{j1}, u^x_{j1}\right] \times \ldots \times \left(\ell^x_{jD}, u^x_{jD}\right] \subseteq B_j$ denote the smallest rectangle that encloses the training data points in node j. Let $\mathsf{leaf}(\boldsymbol{x})$ denote the unique leaf node $j \in \mathsf{leaves}(\mathsf{T})$ such that $\boldsymbol{x} \in B_j$. (Recall that the leaves define a partition of the input space.) For brevity, we will also use the following shortcut notation to label the nodes of the decision tree: label the root node as the empty string ϵ and label $\mathsf{left}(j) = j0$ and $\mathsf{right}(j) = j1$. If each parent-child path is labeled 0 or 1 depending on the outcome of the binary decision, this labeling scheme ensures that the label of each node is the concatenation of the labels along the path from the root till that node. We refer to Figure 2.1 for more intuition on the representation and notation of decision trees.

Once we have a decision tree structure, we also need a rule for predicting the label of a test points given training data. To this end, we will associate each leaf node j with a parameter $\boldsymbol{\theta}_j$ that parametrizes the conditional distribution $p(y|\boldsymbol{x} \in B_j)$. For instance, $\boldsymbol{\theta}_j$ would parametrize the K-dimensional discrete distribution for classification problems and the mean of a Gaussian distribution for regression problems.

2.2.1 Learning decision trees

Learning a decision tree from training data involves two steps namely, learning the tree structure $\mathcal{T}$ and estimating the leaf node parameters $\boldsymbol{\theta}$. Popular decision tree induction algorithms include CART (Breiman et al., 1984) and C4.5 (Quinlan, 1993). While it is possible to learn a deep decision tree until there is an unique data point at each leaf node, it is common to limit the complexity of the decision tree by specifying a hyper-parameter that decides when to stop splitting a node. The most popular strategy is to require a minimum number of samples ($\mathsf{min\_samples\_split}$) at a node before it can be split. (A variant of this strategy is to require that a split leads to a minimum number of samples $\mathsf{min\_samples\_leaf}$ at each leaf node.) Alternative strategies include not splitting a node if all the class labels are identical (for classification problems) or limiting the maximum depth of the tree; however specifying maximum depth is relatively harder to specify in a dataset-agnostic fashion (since deeper and/or unbalanced trees might be preferable for some datasets). Due to its simplicity and robustness, we prefer $\mathsf{min\_samples\_split}$.

We describe a typical decision tree induction algorithm in Algorithms 2.1 and 2.2. The procedure starts with the root node ϵ and recurses down the tree. At node j, $\mathsf{CandidateSplits}_j$ denotes the set of candidate pair of *valid* split dimensions and locations, where a valid split is one where both children are non-empty. In practice, the set of valid split candidates is obtained by sorting the training data independently along each dimension; since the training data takes on only along a finite number of unique values, it is sufficient to consider a single split location (usually the midpoint) for each

of these intervals (as any split location along this midpoint has the same accuracy on the training data). We greedily choose the best split dimension and split location from $\mathsf{CandidateSplits}_j$ by optimizing an appropriate criterion, e.g. information gain or Gini index for classification and reduction in MSE for regression.

Algorithm 2.1 $\mathsf{BuildDecisionTree}(\mathcal{D}_{1:n}, \mathsf{min\_samples\_split})$

Initialize empty tree: $\mathsf{T} = \emptyset$, $\mathsf{leaves}(\mathsf{T}) = \emptyset$, $\boldsymbol{\delta} = \emptyset$, $\boldsymbol{\xi} = \emptyset$
Set $N_\epsilon = \{1, 2, \ldots, n\}$ ▷ *entire dataset is used at root node*
$\mathsf{ProcessBlock}(\epsilon, \mathcal{D}_{N_\epsilon}, \mathsf{min\_samples\_split})$ ▷ *Algorithm 2.2*

Algorithm 2.2 $\mathsf{ProcessBlock}(j, \mathcal{D}_{N_j}, \mathsf{min\_samples\_split})$

Add j to T
if $|N_j| \geq \mathsf{min\_samples\_split}$ **then** ▷ *j is an internal node.*
    Set $\mathsf{CandidateSplits}_j$ to the set of all valid pairs of split dimensions and locations
    Choose best split dimension δ_j and split location ξ_j amongst $\mathsf{CandidateSplits}_j$ by optimizing appropriate criterion ▷ *greedy optimization*
    Set $N_{\mathsf{left}(j)} = \{n \in N_j : \boldsymbol{X}_{n,\delta_j} \leq \xi_j\}$ and $N_{\mathsf{right}(j)} = \{n \in N_j : \boldsymbol{X}_{n,\delta_j} > \xi_j\}$
    $\mathsf{ProcessBlock}(\mathsf{left}(j), \mathcal{D}_{N_{\mathsf{left}(j)}}, \mathsf{min\_samples\_split})$
    $\mathsf{ProcessBlock}(\mathsf{right}(j), \mathcal{D}_{N_{\mathsf{right}(j)}}, \mathsf{min\_samples\_split})$
else ▷ *j is a leaf node*
    Add j to $\mathsf{leaves}(\mathsf{T})$
    Estimate $\boldsymbol{\theta}_j$ using Y_{N_j}

For leaf nodes, we estimate the parameters $\boldsymbol{\theta}_j$ using $\mathcal{D}_{N_j}$. In the simplest case, $\boldsymbol{\theta}_j$ is estimated just using Y_{N_j}, independent of $\boldsymbol{X}_{N_j}$; there exist variants where $\boldsymbol{\theta}_j$ also depends on $\boldsymbol{X}_{N_j}$, however we restrict our attention to the former since it is computationally fast. For classification problems, let c_{jk} denote the number of data points in node j with label k, i.e. $c_{jk} = \sum_{n \in N_j} \mathbb{1}[y_n = k]$; in this case, $\boldsymbol{\theta}_j$ is estimated as

$$\theta_{jk} = \frac{c_{jk} + \alpha}{|N_j| + K\alpha},$$

where we add a small constant α to smooth the empirical histogram of labels in node j and $|N_j|$ (the size of the set N_j) denotes the number of data points in node j. For regression problems, $\boldsymbol{\theta}_j$ is set to the empirical mean of the labels in node j, i.e.

$$\boldsymbol{\theta}_j = \frac{1}{|N_j|} \sum_{n \in N_j} y_n.$$

2.2.2 Prediction with a decision tree

Recall that $\mathsf{leaf}(\boldsymbol{x})$ denotes the unique leaf node $j \in \mathsf{leaves}(\mathsf{T})$ such that $\boldsymbol{x} \in B_j$. Prediction from a tree involves two steps: (i) traversing the decision tree starting from the root node to identify $\mathsf{leaf}(\boldsymbol{x})$ and (ii) returning (a function of) the leaf node parameter $\boldsymbol{\theta}_{\mathsf{leaf}(\boldsymbol{x})}$. For regression, the prediction is the mean $\boldsymbol{\theta}_{\mathsf{leaf}(\boldsymbol{x})}$, whereas for classification,

one can return either the probabilistic prediction $\boldsymbol{\theta}_{\mathsf{leaf}(\boldsymbol{x})}$ or the most likely class label $\mathsf{argmax}_k \boldsymbol{\theta}_{\mathsf{leaf}(\boldsymbol{x}),k}$. The procedure is summarized in Algorithm 2.3.

Algorithm 2.3 $\mathsf{Predict}(\mathcal{T}, \boldsymbol{x})$ (prediction using decision tree)

▷ *Description of prediction using a decision tree given* $\mathcal{T}$ *and* $\boldsymbol{\theta}$
Initialize $j = \epsilon$
while True **do**
  if $j \in \mathsf{leaves}(\mathsf{T})$ **then** ▷ *Reached* $\mathsf{leaf}(\boldsymbol{x})$
    return prediction $\boldsymbol{\theta}_j$
  else
    if $x_{\delta_j} \leq \xi_j$ **then** $j \leftarrow \mathsf{left}(j)$ **else** $j \leftarrow \mathsf{right}(j)$ ▷ *recurse to child where* $\boldsymbol{x}$ *lies*

2.3 Bayesian decision trees

In the previous section, we described a simple tree induction procedure to learn a decision tree and the leaf node parameters. However, a potential drawback is that the greedy induction procedure can over-fit the training data, thereby leading to overconfident predictions on unseen data. Assume that the labels were generated according to a decision tree $\mathcal{T}^*$ (the 'ground truth'). Given finite training data, the greedy learning algorithm returns an estimate $\widehat{\mathcal{T}}$ (a single tree) which does not equal $\mathcal{T}^*$ in general. Specifically, there are two issues: first, there could be multiple decision tree structures that are equally good at explaining the training data; however the induction algorithm returns just a single decision tree. Next, the leaf node parameters are estimated using just the data points at that leaf node; this may lead to poor generalization. Clearly, it would be desirable to represent the uncertainty over decision tree structures and the leaf node parameters.

The Bayesian approach (Bayes and Price, 1763) provides a principled solution to this issue. The Bayesian approach is conceptually very simple. First, we introduce a *prior* over decision trees (e.g. a prior that prefers shallow trees) and the leaf node parameters (e.g. a prior that prefers smaller values for regression or a prior that encourages sparse label distributions for classification). Next, we define a *likelihood* which measures how well a decision tree explains the given training data. Finally, we compute the *posterior distribution* over decision trees and the node parameters using Bayes theorem:

$$\underbrace{p(\mathcal{T}, \boldsymbol{\theta}|Y, \boldsymbol{X})}_{posterior} = \frac{1}{\mathcal{Z}(Y, \boldsymbol{X})} \underbrace{p(Y|\mathcal{T}, \boldsymbol{\theta}, \boldsymbol{X})}_{likelihood} \underbrace{p(\boldsymbol{\theta}|\mathcal{T})p(\mathcal{T}|\boldsymbol{X})}_{prior},$$

$$\mathcal{Z}(Y, \boldsymbol{X}) = \sum_{\mathcal{T}} \int_{\boldsymbol{\theta}} p(Y|\mathcal{T}, \boldsymbol{\theta}, \boldsymbol{X})\ p(\boldsymbol{\theta}|\mathcal{T})\ p(\mathcal{T}|\boldsymbol{X}) d\boldsymbol{\theta},$$

where $\mathcal{Z}(Y, \boldsymbol{X})$ is the so-called *marginal likelihood* of the training data. During prediction,

the predictions of trees are weighted according to the posterior distribution, i.e.,

$$p(y|\boldsymbol{x}) = \sum_{\mathcal{T}} \int_{\boldsymbol{\theta}} p(y|\boldsymbol{x}, \mathcal{T}, \boldsymbol{\theta}) p(\mathcal{T}, \boldsymbol{\theta}|Y, \boldsymbol{X}) d\boldsymbol{\theta}.$$

This process is known as *Bayesian model averaging* (Hoeting et al., 1999) and accounts for the uncertainty in the model (in this case, the model is the decision tree along with the leaf node parameters), unlike the previous approach which predicts just using a single decision tree and set of leaf node parameters. The Bayesian approach allows us to quantify predictive uncertainty (by translating the model uncertainty into predictive uncertainty) which is useful in a variety of applications such as cost-sensitive decision making, reinforcement learning, etc.

The main challenge in the Bayesian approach is the computational complexity. While computing the Bayesian posterior over node parameters is typically straight forward, computing the exact posterior distribution over trees is infeasible for non-trivial problems and in practice, we have to resort to approximations. Specifically, the integral over $\boldsymbol{\theta}$ is typically easy to compute as the likelihood is assumed to belong to the exponential family distribution and the prior over $\boldsymbol{\theta}$ is the corresponding conjugate prior. However, the summation over $\mathcal{T}$ is computationally intractable as there are exponentially many trees. In practice, the posterior is approximated with a finite set of trees as follows:

$$\begin{aligned} p(y|\boldsymbol{x}) &\approx \sum_{s=1}^{S} \overline{w}_s \; p(y|\boldsymbol{x}, \mathcal{T}_s), \\ &= \sum_{s=1}^{S} \overline{w}_s \int_{\boldsymbol{\theta}} p(y|\boldsymbol{x}, \mathcal{T}_s, \boldsymbol{\theta}) p(\boldsymbol{\theta}|Y, \boldsymbol{X}, \mathcal{T}_s) d\boldsymbol{\theta}, \end{aligned} \tag{2.1}$$

where $\sum_s \overline{w}_s = 1$. It is possible to approximate the posterior using standard tools such as Markov chain Monte Carlo (MCMC). Some early examples of Bayesian decision trees are Buntine (1992); Chipman et al. (1998); Denison et al. (1998). Intuitively, these posterior approximations replace the intractable sum over trees with a finite summation by focusing only on the promising trees and ignoring trees whose posterior weights are close to zero. We discuss Bayesian decision tree algorithms in more detail in Chapter 3.

2.4 Ensembles of decision trees

In *ensemble learning*, many 'weak' predictors are combined to obtain a 'powerful' predictor (Dietterich, 2000) that is more accurate than the individual predictors. In the simplest case, the predictions from the ensemble are just a weighted additive combination of the predictions from the individual predictors. While ensemble learning can be combined with any learning algorithm, ensembles of decision trees are very popular since decision trees are computationally fast to train and test. Ensembles of

decision trees often achieve state-of-the-art performance in many supervised learning problems (Caruana and Niculescu-Mizil, 2006). Let $\{\mathcal{T}_m, \boldsymbol{\theta}_m\}_{m=1}^M$ denote an ensemble of trees, where M denotes the number of trees in the ensemble. Let $g(\boldsymbol{x}; \mathcal{T}_m, \boldsymbol{\theta}_m)$ denote the prediction from the m^{th} decision tree for a test data point $\boldsymbol{x}$. (We slightly abuse the notation to allow the prediction to either be a point estimate or a probability distribution or density.) The prediction from an ensemble can be written as

$$g(\boldsymbol{x}; \{\mathcal{T}_m, \boldsymbol{\theta}_m\}_{m=1}^M) = \sum_{m=1}^{M} w_m \; g(\boldsymbol{x}; \mathcal{T}_m, \boldsymbol{\theta}_m). \tag{2.2}$$

Ensembles of decision trees can be broadly classified into two families: *additive/boosted decision trees*, wherein each tree fits the residual not explained by the remainder of the trees, and *random forests*, wherein randomized independent decision trees are grown independently and predictions are averaged to reduce variance. We briefly review these variants below.

2.4.1 Additive decision trees

Boosting is an ensemble learning framework where each predictor is trained to focus on the mistakes of the other predictors. Early boosting algorithms include the AdaBoost algorithm for binary classification proposed by Freund and Schapire (1997) and the *gradient boosted regression trees (GBRT)* algorithm proposed by Friedman (2001) for regression problems. An high-level pseudocode for fitting an ensemble of boosted regression trees is described in Algorithm 2.4. (Note that this is just a high-level pseudocode; it is important to prevent individual trees from overfitting cf. (Friedman, 2002).)

Algorithm 2.4 Pseudocode for learning boosted regression trees

1: Inputs: Training data $(\boldsymbol{X}, Y)$
2: **for** $m = 1 : M$ **do**
3: Compute residual $R_m = Y - \sum_{m'=1}^{m-1} g(\boldsymbol{X}; \mathcal{T}_{m'}, \boldsymbol{\theta}_{m'})$.
4: Learn m^{th} decision tree $\mathcal{T}_m, \boldsymbol{\theta}_m$ using R_m as the targets for $\boldsymbol{X}$ ▷ *Algorithm 2.1*

Note that the decision trees are fit in a serial fashion in Algorithm 2.4. Specifically, we compute the residual, which equals the difference between the targets and the sum of predictions of all previous trees, and use this residual as the target for the m^{th} tree. This depth-first expansion can lead to over-fitting. An alternative is to fit the trees in an iterative breadthwise-expansion scheme, where we fit the root of the M trees first, and subsequently fit the individual trees by expanding them, one node at a time. Examples of iteratively fitted additive regression trees include *additive groves* (Sorokina et al., 2007), *Bayesian additive regression trees (BART)* (Chipman et al., 2010) and greedy regularized forest (Johnson and Zhang, 2013). While the term *boosted decision trees*

usually refers to serial-fitting, the term *additive decision trees* includes both serial-fitting and iterative-fitting.

Chipman et al. (2010) introduced Bayesian additive regression trees (BART), which reduce over-fitting in gradient boosted regression trees using a Bayesian approach. Similar to Bayesian decision trees discussed in Section 2.3, BART introduces priors on the decision trees and leaf node parameters and approximates the posterior over the ensemble $\{\mathcal{T}_m, \boldsymbol{\theta}_m\}_{m=1}^M$ using an MCMC sampler. We discuss BART in more detail in Chapter 4.

Caruana and Niculescu-Mizil (2006) found that boosted decision trees were slightly more accurate than random forests. However, boosted decision trees are more sensitive to label noise. Unlike random forests, the computation across trees cannot be parallelized. Another disadvantage is that additive regression trees do not readily extend to multi-class classification problems.

2.4.2 Random forests

Classic decision tree induction procedures choose the best split dimension and location from all candidate splits at each node by optimizing some suitable quality criterion (e.g. information gain) in a greedy manner. In a random forest, the individual trees are randomized to de-correlate their predictions. The most common strategies for injecting randomness are:

- bootstrap aggregation, more commonly referred to as *bagging* (Breiman, 1996) where each decision tree is trained on a slightly different training dataset, and
- randomly subsampling the set of candidate splits within each node.

The prediction from a random forest is usually an (unweighted) average of the predictions of individual trees:

$$g(\boldsymbol{x}; \{\mathcal{T}_m, \boldsymbol{\theta}_m\}_{m=1}^M) = \sum_{m=1}^{M} \frac{1}{M}\, g(\boldsymbol{x}; \mathcal{T}_m, \boldsymbol{\theta}_m).$$

For classification, it is also possible to use majority voting if the individual trees output discrete class labels instead of probability distributions. While it is common to use uniform weights $w_m = M^{-1}$, the weights can also be optimized, e.g. using *stacking* (Wolpert, 1992).

Geurts et al. (2006) discuss the advantage of random forests over decision trees using the bias-variance tradeoff. Individual decision trees have low bias, but exhibit high variance (as tree induction algorithms produce different trees on slightly different versions of the dataset.) In a random forest, the individual trees are randomized in order to decorrelate their predictions. The randomization scheme may slightly increase the bias

of individual trees in the forest. (For instance, if each tree of the forest is trained on a random subset of the training dataset, the individual trees may have lower accuracy than the best possible decision tree.) However, the variance of the forest is much lower than the variance of the individual trees,[1] which more than compensates for the slight increase in bias, thereby leading to a better predictor. Dietterich (2000) discusses three fundamental reasons why an ensemble might outperform a single classifier. The first reason is statistical: given finite training data, many hypotheses may be equally good on the training data. By combining predictions from multiple good predictors, an ensemble reduces the risk of choosing the wrong hypothesis. The second reason is computational: in cases where the training algorithm is prone to local optima, the ensemble combines the results from multiple random searches and may provide a better approximation to the true unknown function. The third reason is representational: while decision trees can represent any function in principle, the effective hypothesis space is limited by the greedy training algorithm. An ensemble is capable of representing weighted combinations of trees, which increases its effective representational power while training on finite data using a greedy local search.

Two popular random forest variants are *Breiman-RF* (Breiman, 2001) and *Extremely randomized trees (ERT)* (Geurts et al., 2006). Breiman-RF uses bagging and furthermore, at each node, a random k-dimensional subset of the original D features is sampled. ERT chooses a k dimensional subset of the features and then chooses one split location each for the k features randomly (unlike Breiman-RF which considers all possible split locations along a dimension). Unlike Breiman-RF, ERT does not use bagging.

As we will see later, random forests are better-suited than boosted decision trees for different settings such as binary classification, multi-class classification, regression, etc. Random forests are very easy to implement as they only involve a minor change of the decision tree pseudocode. For instance, bagging just requires setting N_ϵ in Algorithm 2.1 to a bootstrap sample instead of the full dataset. Similarly, random split sampling just requires setting $\mathsf{CandidateSplits}_j$ to a subset of the valid splits in Algorithm 2.2. Another advantage is that the individual trees can be trained in parallel since they do not interact with each other. Fernández-Delgado et al. (2014) compared a suite of machine learning algorithms on a variety of datasets and found that random forests consistently rank among the top-performing algorithms. Due to these advantages, random forests remain one of the most popular black-box prediction algorithms. We refer the reader to (Criminisi et al., 2012) for an excellent review of random forests and other extensions such as density estimation, manifold learning and semi-supervised learning.

While the random forest framework is very powerful, it has a couple of disadvantages. First, random forests do not quantify predictive uncertainty in a principled way. Specifically, methods such as Gaussian processes have the appealing property that uncertainty

[1] Specifically, the variance of a forest with M trees is M times lower than the variance of individual trees.

increases as we move farther away from the training data. However, predictions from a random forest can be over-confident even in regions where training data has not been observed. The main reason for this difference is that Gaussian processes are probabilistic whereas random forests are not. In a probabilistic framework, we first posit a prior that represents our uncertainty about the parameters of the underlying function, and next posit a likelihood function that measures how well the parameters explain the observed training data. Finally, we compute the predictive posterior using Bayes theorem. The observed data constrains the function by down-weighting unlikely parameters; hence the predictive posterior is less uncertain in regions close to the observed training data. However, the observed training data does not constrain the function in regions far away from the training data, hence the predictive posterior reduces to the prior distribution and exhibits higher uncertainty (as expected) in regions far from the observed training data.

Another disadvantage of random forests is that they are not well suited for *incremental or online learning* setting where we observe new data points on-the-fly (unlike the *batch learning* setting where the training dataset does not grow with time.) Large-scale machine learning systems for streaming data are often trained using stochastic gradient algorithms. However, random forests with hard splits are not amenable to gradient based updates. Since it is difficult to undo splits in decision trees, current online random forests wait until they have seen sufficient amount of data to confidently decide the split. Hence, they are very data inefficient compared to the corresponding batch random forest. We propose a novel variant of random forests that addresses both of these issues in Chapters 5 and 6.

2.5 Bayesian model averaging vs model combination

We have discussed several algorithms so far. In this section, we discuss the connections between the different algorithms and clarify the differences between seemingly similar approaches. The connections between the algorithms are summarized in Figure 2.2.

We start with decision trees on the top left corner of Figure 2.2. Red lines indicate additive combination (or boosting), where the components are fit jointly in a serial fashion; for instance additive trees combine decision trees. Green lines indicate randomized averaging where multiple randomized versions of the underlying predictor are trained in parallel and their predictions are averaged; for instance random forests average predictions from multiple randomized decision trees. It is possible to apply combine bagging and boosting; Pavlov et al. (2010) proposed *BagBoo*, where multiple randomized versions of boosted decision trees are fit in parallel and averaged. Blue lines indicate Bayesian treatment of the decision tree structures and the leaf node parameters. As the name suggests, Bayesian decision trees perform BMA over decision trees, whereas BART performs Bayesian inference over additive combination of decision trees.

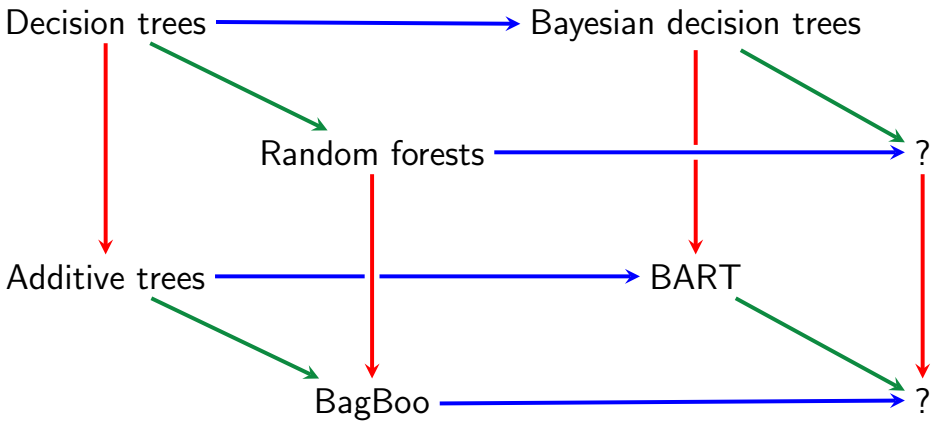


Figure 2.2: Comparison of different approaches: blue horizontal lines denote Bayesian version, red vertical lines denote additive combination, and green lines denote an ensemble combination of randomized predictors. The top-down particle filtering algorithm described in Chapter 3 is a novel Bayesian decision tree variant (top right node). The PG-BART algorithm described in Chapter 4 is a novel variant of BART (and Bayesian decision trees). Mondrian forests, described in Chapters 5 and 6, are a novel hybrid variant of random forests, where we perform Bayesian inference over node parameters in each tree but combine the trees in a non-Bayesian fashion. Furthermore, in Mondrian forests, we restrict splits to the range of observed training data, which allows us to represent uncertainty about the partition structure beyond the range of training data.

Equation (2.1) describing BMA in Bayesian decision trees and equation (2.2) describing the prediction of an ensemble appear to be strikingly similar. It seems tempting to interpret BMA as an ensemble algorithm, however the goals of BMA are quite different. Domingos (2000) interpreted BMA as an ensemble method and claimed that BMA is prone to over-fitting, however Minka (2000) showed that the ensemble interpretation of BMA is incorrect. Ensembles perform *model combination* and hence their hypothesis class is bigger. On the other hand, BMA in Bayesian decision trees assumes that the data was generated by a decision tree and accounts for the uncertainty over trees due to the fact that we observe only finite training data. On finite data, BMA performs *soft model selection* instead of model combination. In fact, in the limit of infinite data, the Bayesian posterior over decision trees would converge to a single tree and only one of the weights in (2.1) would be non-zero. If the data was generated by an ensemble of trees instead of a single decision trees, Bayesian decision trees would not be appropriate as the assumptions of BMA are violated. (We refer to (Minka, 2000) for a simple illustration of the difference between model combination and BMA. Clarke (2003) provides a comparison between BMA and ensemble weighting when the model approximation error cannot be ignored.) The correct solution is to assume that the data was generated by an additive combination of trees and perform BMA over additive combinations of trees instead of BMA over decision trees. In fact, this is the approach taken in BART (Chipman et al., 2010), as we will see in Chapter 4. It is possible to interpolate between BMA over decision trees (where each tree is weighted by its posterior probability) and random forests (where the trees are weighted uniformly).

Quadrianto and Ghahramani (2015) propose to use *power likelihood* which enables interpolation between Bayesian model averaging and model combination.

It is important to note that the term 'Bayesian' in Bayesian decision trees refers to Bayesian inference over both the decision tree structures *and* the leaf node parameters. In Chapters 5 and 6, we discuss *Mondrian forests*, where we perform Bayesian inference over leaf node parameters but combine the decision trees in a non-Bayesian fashion using model combination instead of BMA. Furthermore, in Mondrian forests, we restrict splits to the range of observed training data, which allows us to represent uncertainty about the partition structure beyond the range of training data. Mondrian forests use a hierarchy over the node parameters, conditional on the tree structure, and use Bayesian inference within each tree independently to obtain probabilistic predictions. Since splits are confined to bounding boxes, we can represent uncertainty in the tree structure in regions far away from training data; hierarchical Bayesian inference over node parameters ensures that we efficiently make use of observed training data. Hence, Mondrian forests can produce principled uncertainty estimates. Taddy et al. (2015) propose *Bayesian and empirical Bayesian forests*, where the bootstrap in random forest is replaced by the Bayesian bootstrap (Rubin et al., 1981); however, they do not perform Bayesian inference over the decision trees and leaf node parameters. The real challenge of Bayesian inference in trees and ensembles is Bayesian inference over (exponentially many) decision trees, hence we do not refer to a decision tree (or forest) algorithm as 'Bayesian' unless it learns the posterior over decision trees. Bayesian inference over tree structures is computationally challenging the in incremental/online learning setting; Mondrian forests do not perform Bayesian inference over tree structures, which is part of the reason why they are computationally attractive in this setting.

Decision trees are also reminiscent of so-called mixture of experts (Jacobs et al., 1991), which learn multiple predictors (experts) and additionally learn to use a different predictor for different subsets of data. Decision trees with hard splits learn both the partitioning tree structure as well as the predictors at the leaf nodes, which are the experts in this case. It is also possible to replace the hard splits in a decision tree with soft routing functions that route each data point to the left or right stochastically. Hierarchical mixture of experts (HMEs) (Jordan and Jacobs, 1994) parametrize the routing function using sigmoid functions, and learn the parameters using expectation-maximization algorithm. One issue with the soft routing operation is that a data point needs to be propagated to every leaf, which destroys the computational advantage for deep trees. However, the sigmoid is differentiable which makes it amenable to gradient based end-to-end training. It would be interesting to develop efficient Bayesian versions of HMEs; however, we restrict our attention to decision trees with hard axis-aligned splits in the rest of the thesis.

Chapter 3

SMC for Bayesian decision trees

3.1 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 and McCulloch (2000), and more recently revisited by Wu et al. (2007), Taddy et al. (2011) and Anagnostopoulos and 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 decision 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.

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 chapter is organized as follows: we begin by describing the Bayesian decision tree model precisely in Section 3.2, and then describe the SMC algorithm in detail in Section 3.3. Through a series of empirical tests, we demonstrate in Section 3.4 that this approach is fast and produces good approximations. We conclude in Section 3.5 with a discussion comparing this approach with existing ones in the Bayesian setting, and point towards future avenues.

3.2 Model

3.2.1 Problem setup

We assume that the training data consist of N i.i.d. samples $\boldsymbol{X} = \{\boldsymbol{x}_n\}_{n=1}^N$, where $\boldsymbol{x}_n \in \mathbb{R}^D$, along with corresponding labels $\boldsymbol{Y} = \{y_n\}_{n=1}^N$, where $y_n \in \{1, \ldots, K\}$. We focus only on the multi-class classification task here, although the extension to regression is fairly straightforward. We refer to Section 2.2 and Figure 2.1 for a review of decision trees and our notation.

3.2.2 Likelihood model

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

$$\begin{aligned} p(Y, \mathcal{T} \mid \boldsymbol{X}) &= p(\mathcal{T} \mid \boldsymbol{X})\, p(Y \mid \mathcal{T}, \boldsymbol{X}) \\ &= p(\mathcal{T} \mid \boldsymbol{X}) \prod_{j \in \mathsf{leaves}(\mathsf{T})} \ell(Y_{N_j} | \boldsymbol{X}_{N_j}) \end{aligned} \tag{3.1}$$

where ℓ denotes a likelihood, defined below.

In this chapter, 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,

$$\ell(Y_{N_j} | \boldsymbol{X}_{N_j}) = \frac{\Gamma(\alpha)}{\Gamma(\frac{\alpha}{K})^K} \frac{\prod_{k=1}^K \Gamma(c_{jk} + \frac{\alpha}{K})}{\Gamma(\sum_{k=1}^K c_{jk} + \alpha)}, \tag{3.2}$$

where c_{jk} denotes the number of labels $y_n = k$ among those $n \in N_j$ and α is the concentration parameter of the symmetric Dirichlet prior. Generalizations to other likelihood functions based on conjugate pairs of exponential families are straightforward.

3.2.3 Sequential generative process for trees

The final piece of the model is the prior density $p(\mathcal{T}|\boldsymbol{X})$ 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 $\boldsymbol{X}$ (see Section 3.5 for a discussion of this dependence on $\boldsymbol{X}$ 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 and independently.

We now describe the generative process more precisely in terms of a Markov chain capturing the construction of a decision tree in stages, beginning with the trivial tree $\mathsf{T}_{(0)} = \{\epsilon\}$ containing only the root node, and sampling a sequence of *partial trees*. Let $E_{(t)}$ denotes the ordered set containing the list of nodes *eligible for expansion* at stage t (These are the leaf nodes from $\mathsf{T}_{(t-1)}$ that have not been expanded yet.) At each stage t, $\mathsf{T}_{(t)}$ is produced from $\mathsf{T}_{(t-1)}$ by choosing one eligible node in $E_{(t)}$ 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 $\mathcal{T}_{(t)} = (\mathsf{T}_{(t)}, \boldsymbol{\delta}_{(t)}, \boldsymbol{\xi}_{(t)})$ once we define the

associated cut functions $\boldsymbol{\delta}_{(t)}$ and $\boldsymbol{\xi}_{(t)}$. We will use this Markov chain in Section 3.3 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 j be the node chosen at some stage of the generative process. If the input vectors $\boldsymbol{X}_{N_j}$ are all identical, then the node stops and becomes a leaf. (Chipman et al. (1998) chose this rule because no choice of cut to the block B_j would result in both children containing at least one input vector.) Otherwise, let $\mathcal{V}_j$ be the set of dimensions along which $\boldsymbol{X}_{N_j}$ varies, and let $e^x_{j,d} = [\ell^x_{j,d}, u^x_{j,d}]$ be the extent of the input vectors along dimension $d \in \mathcal{V}_j$. (See last subfigure of Figure 2.1.) Under the Chipman et al. model, the probability that node j is split is

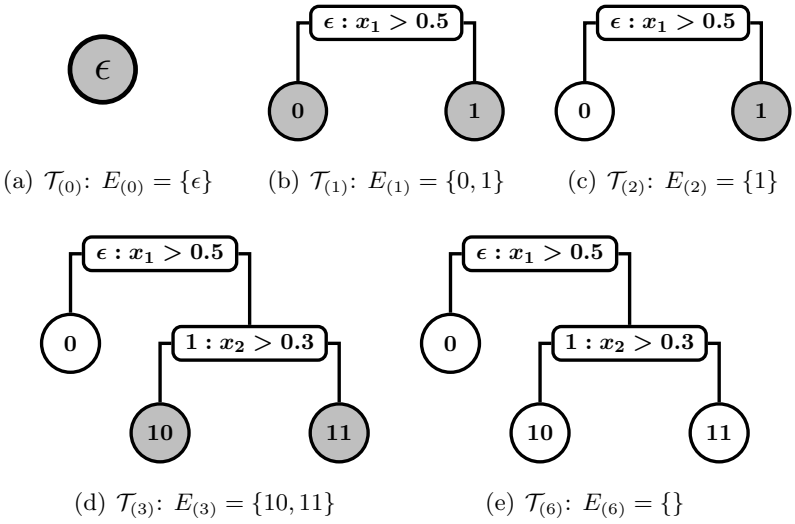
$$\frac{\alpha_s}{(1+\mathsf{depth}(j))^{\beta_s}}\,, \qquad \alpha_s \in (0,1),\ \beta_s \in [0,\infty), \tag{3.3}$$

where $\mathsf{depth}(j)$ 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 j is in the tree the less likely it will be cut. If j is cut, the dimension δ_j and then location ξ_j of the cut are sampled uniformly from $\mathcal{V}_j$ and $\mathbf{e}^x_{j,\delta_j}$, respectively. Note that the choice for the support of the distribution over cut dimensions and locations are such that both children of j 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. Figure 3.1 presents a cartoon of the sequential generative process.

To complete the generative model, we define $\mathsf{T} = \mathsf{T}_\eta$, $\boldsymbol{\delta} = \boldsymbol{\delta}_\eta$ and $\boldsymbol{\xi} = \boldsymbol{\xi}_\eta$, 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 j 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 $\mathcal{T} = (\mathsf{T}, \boldsymbol{\delta}, \boldsymbol{\xi})$ can now be expressed as

$$\begin{aligned} p(\mathsf{T}, \boldsymbol{\delta}, \boldsymbol{\xi} | \boldsymbol{X}) = &\prod_{j \in \mathsf{leaves}(\mathsf{T})} \left(1 - \frac{\alpha_s}{(1+\mathsf{depth}(j))^{\beta_s}}\right)^{\mathbb{1}(|\mathcal{V}_j|>0)} \\ &\times \prod_{j \in \mathsf{T} \setminus \mathsf{leaves}(\mathsf{T})} \frac{\alpha_s}{(1+\mathsf{depth}(j))^{\beta_s}} \frac{1}{|\mathcal{V}_j|} \frac{1}{|\mathbf{e}^x_{j,\delta_j}|}\,. \end{aligned} \tag{3.4}$$

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 and variance of the corresponding SMC algorithm.



(a) $\mathcal{T}_{(0)}$: $E_{(0)} = \{\epsilon\}$ (b) $\mathcal{T}_{(1)}$: $E_{(1)} = \{0, 1\}$ (c) $\mathcal{T}_{(2)}$: $E_{(2)} = \{1\}$

(d) $\mathcal{T}_{(3)}$: $E_{(3)} = \{10, 11\}$ (e) $\mathcal{T}_{(6)}$: $E_{(6)} = \{\}$

Figure 3.1: Sequential generative process for decision trees: Nodes *eligible for expansion* are denoted by the ordered set E and shaded in gray. In every iteration, the first element of E, say j, is popped and is stochastically assigned to be an internal node or a leaf node with probability given by (3.3) At iteration 0, we start with the empty tree and $E = \{\epsilon\}$. At iteration 1, we pop ϵ from E and assign it to be an internal node with split dimension $\delta_\epsilon = 1$ and split location $\xi_\epsilon = 0.5$ and append the child nodes 0 and 1 to E. At iteration 2, we pop 0 from E and set it to a leaf node. At iteration 3, we pop 1 from E and set it to an internal node, split dimension $\delta_1 = 2$ and threshold $\xi_1 = 0.3$ and append the child nodes 10 and 11 to E. At iterations 4 and 5 (not shown), we pop nodes 10 and 11 respectively and assign them to be leaf nodes. At iteration 6, $E = \{\}$ and the process terminates. By arranging the random variables ρ and δ, ξ (if applicable) for each node in the order of expansion, the tree can be encoded as a sequence.

3.3 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 $(\mathsf{T}, \boldsymbol{\delta}, \boldsymbol{\xi})$ given the labeled training data $(\boldsymbol{X}, Y)$. (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 t, the particles approximate a modified posterior distribution where the prior on $(\mathsf{T}, \boldsymbol{\delta}, \boldsymbol{\xi})$ is replaced by the distribution of $(\mathsf{T}_{(t)}, \boldsymbol{\delta}_{(t)}, \boldsymbol{\xi}_{(t)})$, i.e., the process truncated at stage t.

Recall that $E_{(t)}$ denotes the ordered set of unstopped leaves at stage t, all of which are eligible for expansion. We refer to these nodes as *candidates* as they are eligible for expansion. An important freedom we have in our SMC algorithm is the choice of which *candidate* leaf, or set $\mathcal{C}_{(t)} \subseteq E_{(t)}$ 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 $\mathcal{C}_{(t)} \subseteq E_{(t)}$. (Multiple candidates are expanded or stopped in turn, independently.) This rule can be a function of $(\boldsymbol{X}, Y)$ and the state of the current particle, as the correctness of resulting approximation is unaffected. We evaluate two choices in experiments: first, the rule $\mathcal{C}_{(t)} = E_{(t)}$ where we consider expanding all eligible nodes; and second, the rule where $\mathcal{C}_{(t)}$ contains a single node chosen in a breadth-first (i.e., oldest first) manner from $E_{(t)}$. (We consider only breadth-first expansion as it closely resembles top-down tree induction algorithms and allows us to interpret (t) as a surrogate for complexity of the tree.)

We may now define the sequence $(\mathbb{P}^Y_{(t)})$ of **target distributions**. Recall the sequential process defined in Section 3.2. If the generative process for the decision tree has not completed by stage t, the process has generated $(\mathsf{T}_{(t)}, \boldsymbol{\delta}_{(t)}, \boldsymbol{\xi}_{(t)})$ along with $E_{(t)}$, capturing which leaves in $\mathsf{T}_{(t)}$ have been considered for expansion in previous stages already and which have not. Let $\mathcal{T}_{(t)} = (\mathsf{T}_{(t)}, \boldsymbol{\delta}_{(t)}, \boldsymbol{\xi}_{(t)}, E_{(t)})$ be the variables generated on stage t, and write $\mathbb{P}$ for the prior distribution on the sequence $(\mathcal{T}_{(t)})$. We construct the target distribution $\mathbb{P}^Y_{(t)}$ as follows: Given $\mathcal{T}_{(t)}$, we generate labels Y' with likelihood $p(Y'|\mathcal{T}_{(t)}, \boldsymbol{X})$, i.e., as if $(\mathsf{T}_{(t)}, \boldsymbol{\delta}_{(t)}, \boldsymbol{\xi}_{(t)})$ were the complete decision tree. We then define $\mathbb{P}^Y_{(t)}$ to be the conditional distribution of $\mathcal{T}_{(t)}$ given $Y' = Y$. That is, $\mathbb{P}^Y_{(t)}$ is the posterior with a truncated prior.

In order to complete the description of our SMC method, we must define **proposal kernels** $(\mathbb{Q}_{(t)})$ that sample approximations for the tth stage given values for the $(t-1)$th stage. As with our choice of $\mathcal{C}_{(t)}$, we have quite a bit of freedom. In particular, the proposals can depend on the training data $(\boldsymbol{X}, Y)$. An obvious choice is to take $\mathbb{Q}_{(t)}$ to be the conditional distribution of $\mathcal{T}_{(t)}$ given $\mathcal{T}_{(t-1)}$ under the prior, i.e., setting $\mathbb{Q}_{(t)}(\mathcal{T}_{(t)} \,|\, \mathcal{T}_{(t-1)}) = \mathbb{P}(\mathcal{T}_{(t)} \,|\, \mathcal{T}_{(t-1)})$.

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,

$$\mathbb{Q}_{(t)}(\mathcal{T}_{(t)} \mid \mathcal{T}_{(t-1)}) = \mathbb{P}^{Y}_{(t)}(\mathcal{T}_{(t)} \mid \mathcal{T}_{(t-1)}), \tag{3.5}$$

is called the **(one-step) optimal proposal** kernel because it would be the optimal kernel assuming that the tth stage were the final stage. We return to discuss this kernel in Section 3.3.1. 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 ξ. Recall that, in the prior, $\xi_{(t),j}$ is chosen uniformly from the interval $\mathbf{e}^{x}_{j,\delta_j}$. This ignores the empirical distribution given by the input data $\boldsymbol{X}_{N_j}$ in the partition. We can account for this by first choosing, uniformly at random, a pair of adjacent data points along feature dimension $\delta_{(t),j}$, and then sampling a cut $\xi_{(t),j}$ uniformly from the interval between these two data points.

The pseudocode for our proposed SMC algorithm is given in Algorithm 3.1. Note that the SMC framework only requires us to compute the density of $\mathcal{T}_{(t)}$ 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 $\boldsymbol{X}$, 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 $\mathcal{C}_{(t)}$ considered at each stage is a deterministic function of $\mathcal{T}_{(t)}$, the path $(\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_{(t-1)})$ taken is a deterministic function of $\mathcal{T}_{(t)}$. 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 $\mathcal{C}_{(t)}$, and have the same support as $\mathbb{P}$. These properties justify equations (3.6) and (3.7) in Algorithm 3.1.

3.3.1 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 $\mathbb{Q}_{(t)}$ is given by $\mathbb{Q}_{(t)}(\mathcal{T}_{(t)} \mid \mathcal{T}_{(t-1)}) = \mathbb{P}^{Y}_{(t)}(\mathcal{T}_{(t)} \mid \mathcal{T}_{(t-1)})$. To begin, we note that, conditionally on $\mathcal{T}_{(t-1)}$ and Y, the subtrees rooted at each node $j \in \mathcal{C}_{(t-1)}$ are independent. This follows from the fact that the likelihood of Y given $\mathcal{T}_{(t)}$ factorizes over the leaves. Thus, the proposal's probability

Algorithm 3.1 SMC for Bayesian decision tree learning

Inputs: Training data $(\boldsymbol{X}, Y)$, Number of particles C
Initialize: $\mathsf{T}_{(0)}(c) = E_{(0)}(c) = \{\epsilon\}$
$\quad\boldsymbol{\delta}_{(0)}(c) = \boldsymbol{\xi}_{(0)}(c) = \emptyset$
$\quad w_{(0)}(c) = p(Y|\mathcal{T}_{(0)}(c))$
$\quad W_{(0)} = \sum_c w_{(0)}(c)$
for $t = 1 :$ MAX-STAGES **do**
  for $c = 1 : C$ **do**
    Sample $\mathcal{T}_{(t)}(c)$ from $\mathbb{Q}_{(t)}(\cdot \mid \mathcal{T}_{(t-1)}(c))$
    where $\mathcal{T}_{(t)}(c) := (\mathsf{T}_{(t)}(c), \boldsymbol{\delta}_{(t)}(c), \boldsymbol{\xi}_{(t)}(c), E_{(t)}(c))$
    Update weights: (Here $\mathbb{P}, \mathbb{Q}_{(t)}$ denote their densities.)

$$w_{(t)}(c) = \frac{\mathbb{P}(\mathcal{T}_{(t)}(c))\, p(Y \mid \mathcal{T}_{(t)}(c), \boldsymbol{X})}{\mathbb{Q}_{(t)}(\mathcal{T}_{(t)}(c) \mid \mathcal{T}_{(t-1)}(c))\, \mathbb{P}(\mathcal{T}_{(t-1)}(c))} \tag{3.6}$$

$$= w_{(t-1)}(c) \frac{\mathbb{P}(\mathcal{T}_{(t)}(c) \mid \mathcal{T}_{(t-1)}(c))}{\mathbb{Q}_{(t)}(\mathcal{T}_{(t)}(c) \mid \mathcal{T}_{(t-1)}(c))} \frac{p(Y \mid \mathcal{T}_{(t)}(c), \boldsymbol{X})}{p(Y \mid \mathcal{T}_{(t-1)}(c), \boldsymbol{X})} \tag{3.7}$$

    Compute normalization: $W_{(t)} = \sum_c w_{(t)}(c)$
    Normalize weights: $(\forall c)\, \overline{w}_{(t)}(c) = w_{(t)}(c)/W_{(t)}$
    if $\left(\sum_c (\overline{w}_{(t)}(c))^2\right)^{-1} <$ ESS-THRESHOLD **then**
      $(\forall c)$ Resample indices a_c from $\sum_{c'} \overline{w}_{(t)}(c')\delta_{c'}$
      $(\forall c)\ \mathcal{T}_{(t)}(c) \leftarrow \mathcal{T}_{(t)}(a_c)$; $w_{(t)}(c) \leftarrow W_{(t)}/C$
    if $(\forall c)\, E_{(t)}(c) = \emptyset$ **then**
      exit for loop
return Estimated marginal probability $W_{(t)}/C$ and weighted samples $\{w_{(t)}(c), \mathsf{T}_{(t)}(c), \boldsymbol{\delta}_{(t)}(c), \boldsymbol{\xi}_{(t)}(c)\}_{c=1}^{C}$.

density is

$$\mathbb{Q}_{(t)}(\mathcal{T}_{(t)}|\mathcal{T}_{(t-1)}) = \prod_{j \in \mathcal{C}_{(t-1)}} Q_{(t)}(\rho_{(t),j}, \delta_{(t),j}, \xi_{(t),j}), \tag{3.8}$$

where $Q_{(t)}$ is the probability density of the cuts at node j under $\mathbb{Q}_{(t)}$, and $\rho_{(t),j}$ denotes whether the node was split or not. On the event we split a node $j \in \mathcal{C}_{(t-1)}$, if we condition further on $\delta_{(t),j}$ and $\rho_{(t),j}$, we note that the conditional likelihood of Y_{N_j}, when viewed as a function of the split $\xi_{(t),j}$, is piecewise constant, and in particular, only changes when the split crosses an example.[1] 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

[1] We implement this sampling step efficiently as follows: first we sort the data points a node along each dimension independently. Note that the conditional likelihood Y_{N_j} changes only when the split crosses an example since the label counts are equal otherwise. Hence, we can compute the likelihood for all valid split points along a dimension, with a linear scan of the data points in sorted (e.g. ascending) order, updating counts only when we cross a data point.

algebra shows that

$$Q_{(t)}(\rho_{(t),j} = \mathsf{stop}) \propto \left(1 - \frac{\alpha_s}{(1+\mathsf{depth}(j))^{\beta_s}}\right) \ell(Y_{N_j}|\boldsymbol{X}_{N_j}),$$

$$Q_{(t)}(\rho_{(t),j} = \mathsf{split}, \delta_{(t),j}, \xi_{(t),j}) \propto \frac{\alpha_s}{(1+\mathsf{depth}(j))^{\beta_s}} \frac{1}{|\mathcal{V}_j|} \frac{1}{|\mathbf{e}^x_{j,\delta_{(t),j}}|} \times \prod_{j'=j0,j1} \ell(Y_{N_{j'}}|\boldsymbol{X}_{N_{j'}}).$$

3.3.2 Computational complexity

Let U_d denote the number of unique values in dimension d, N_j denote the number of training data points at node j and $\eta(c)$ denote the number of nodes in particle c. For all the SMC algorithms, the space complexity is $\mathcal{O}(CN)+\mathcal{O}(\sum_d U_d)+\mathcal{O}(\sum_c \eta(c))$. The time complexity for *prior* and *empirical* proposals is $\mathcal{O}(DN \log N)+C\sum_j \mathcal{O}(2D \log N_j + N_j)$, where $\mathcal{O}(DN \log N)$ corresponds to pre-computation time to sort entire dataset along each dimension independently and $C\sum_j \mathcal{O}(2D \log N_j + N_j)$ corresponds to logarithmic time to find the min and max along each dimension, and time for a linear scan along the data in a node to compute the label counts and assign data points to left or right child. The time complexity for the *optimal* proposal $C\sum_j \big(D\mathcal{O}(N_j \log N_j) + N_j\big)$, where the first term corresponds to the time to sort the data in each node independently along every dimension and the last term corresponds to time for linear scan to assign data points to left or right child (once a split has been sampled). The *optimal* proposal typically requires higher computational cost per particle, but fewer number of particles than the *prior* and *empirical* proposals.

3.4 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 and 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.[2] Our experiments were run on a cluster with machines of similar processing power.

3.4.1 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 C particles, we use an effective sample size (ESS) threshold of $C/10$ for resampling, and set the maximum number of stages to 5000 (although the algorithms never reached this number).

3.4.1.1 Proposal choice and node expansion

We consider the SMC algorithm proposed in Section 3.3 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 $\mathcal{C}_{(t)}$, i.e., the list of nodes considered for expansion at stage t: (i) *node-wise expansion*, where a single node is considered for expansion per stage (i.e., $\mathcal{C}_{(t)}$ is a singleton chosen deterministically from eligible nodes $E_{(t)}$), and (ii) *layer-wise expansion*, where all nodes at a particular depth are considered for expansion simultaneously (i.e., $\mathcal{C}_{(t)} = E_{(t)}$). For *node-wise* expansion, we evaluate two strategies for selecting the node deterministically from $\mathcal{C}_{(t)}$: (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 and accuracy on test data as a function of runtime and of the number of particles. The times reported do not account for prediction time. We average the numbers over 10 random initializations and report standard deviations. The results for test log predictive probability and test accuracy are shown in Figures 3.2 and 3.3 respectively. For concreteness, we analyze the trends with respect to test predictive probability, however test accuracy exhibits similar trends. 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

[2]The scripts can be downloaded from http://www.gatsby.ucl.ac.uk/~balaji/treesmc/.

'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 chapter.

The plots on the right side of Figure 3.2 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 3.3.2 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 ($C = 10$).

3.4.1.2 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[3] 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 3.4. 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

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

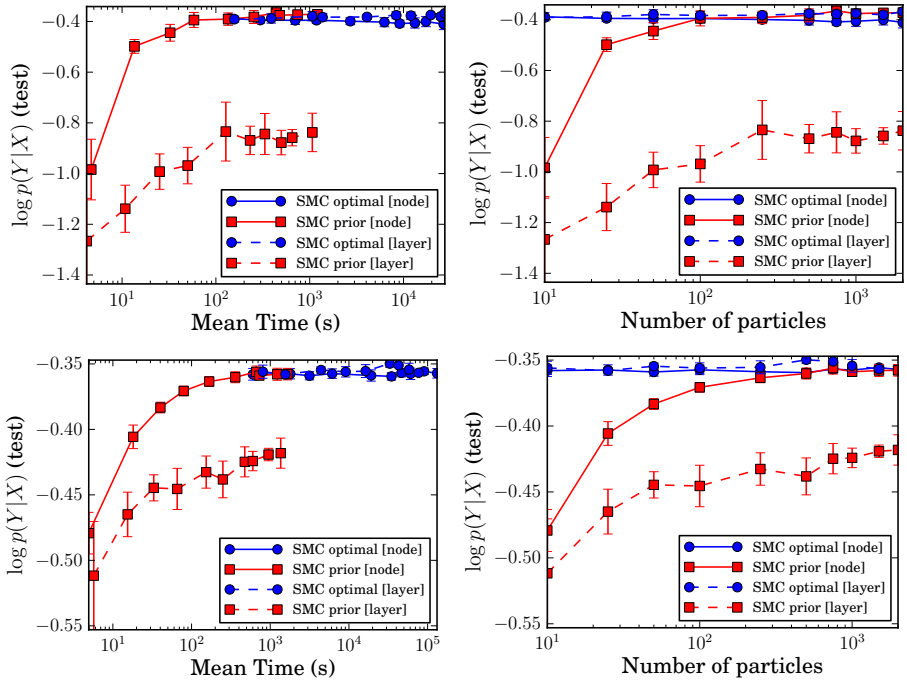


Figure 3.2: Results on *pen-digits* (top), and *magic-04* (bottom). Left column plots test $\log p(y|\boldsymbol{x})$ vs runtime, while right column plots test $\log p(y|\boldsymbol{x})$ vs number of particles. The blue circles and red squares represent *optimal* and *prior* proposals respectively. The solid and dashed lines represent *node-wise* and *layer-wise* proposals respectively.

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.

3.4.1.3 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 performance 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). The results on *pen-digits* and *magic-04* datasets are shown in Figure 3.5. We observe that (i) the test performance drops sharply if we use fewer than 100 particles per island and (ii) when $C/I \geq 100$, the choices of $I \in [5, 100]$ outperform $I = 1$. Since the islands are independent, the computation across islands is *embarrassingly parallelizable*. The island approach also bears similarities to *random forests* (Breiman, 2001), where multiple randomized trees are averaged to reduce variance. However, note that all the islands operate on the entire dataset unlike random forests, where each tree is trained on a *bootstrap* sample of the original dataset. Averaging over multiple islands also improves robustness to model misspecification; see Section 3.4.2 for a related discussion.

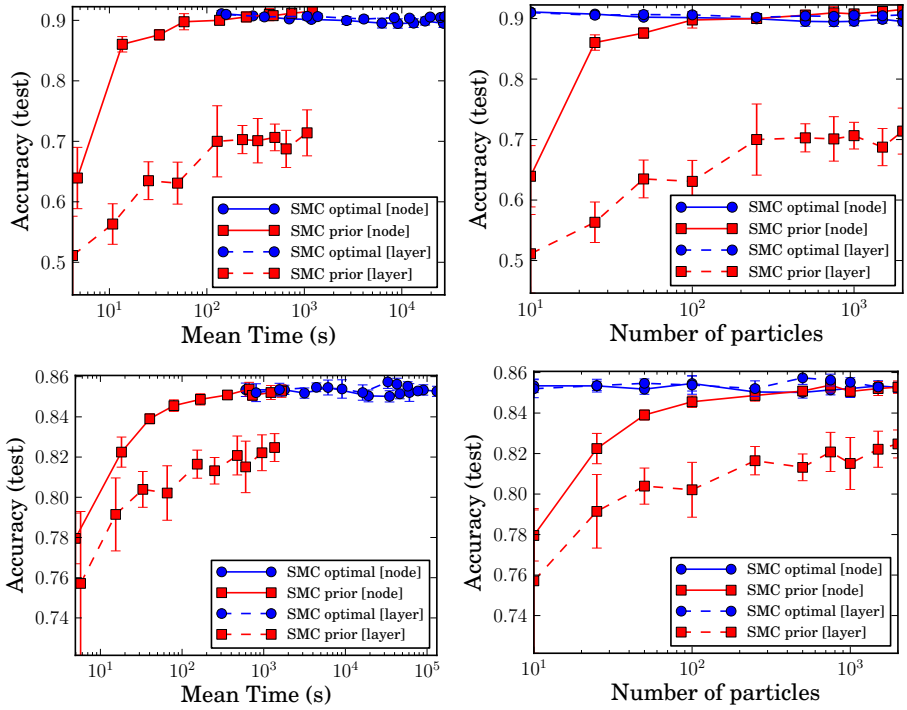


Figure 3.3: Results on *pen-digits* (top), and *magic-04* (bottom). Left column plots test accuracy vs runtime, while right column plots test accuracy vs number of particles. The blue circles and red squares represent *optimal* and *prior* proposals respectively. The solid and dashed lines represent *node-wise* and *layer-wise* proposals respectively.

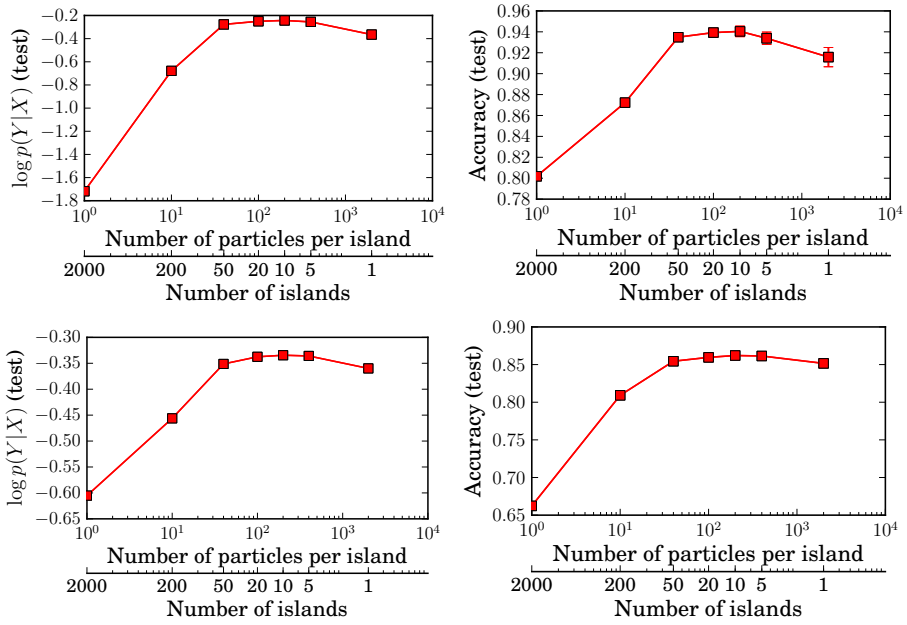


Figure 3.5: Results on *pen-digits* dataset (top row) and *magic-04* dataset (bottom row): Test $\log p(y|\boldsymbol{x})$ (left) and accuracy (right) vs I and C/I for fixed $C = 2000$.

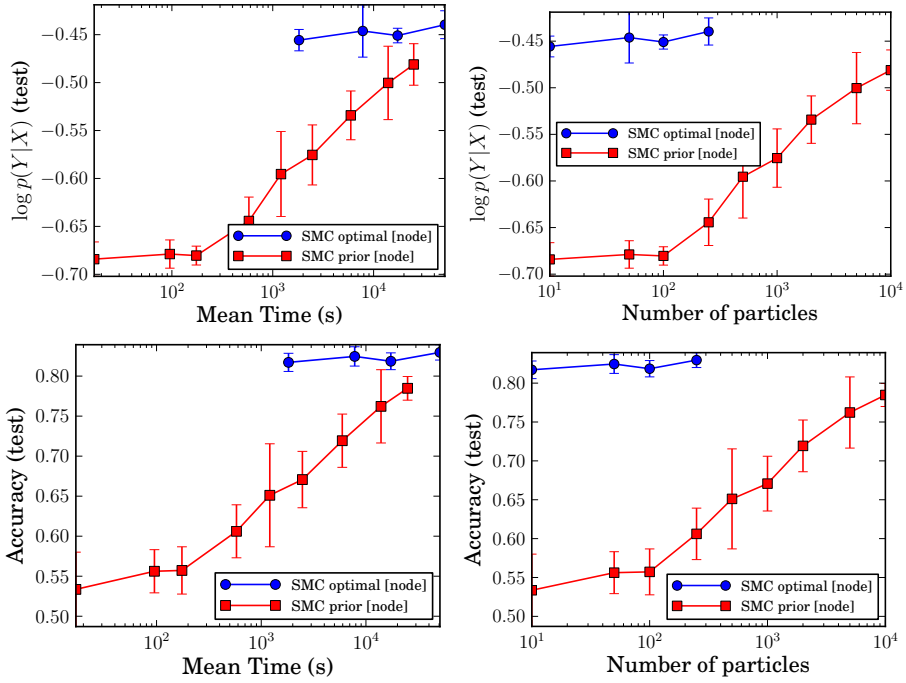


Figure 3.4: Results on *madelon* dataset: The top and bottom rows display $\log p(y|\boldsymbol{x})$ and accuracy on the test data against runtime (left) and the number of particles (right) respectively. The blue circles and red squares represent *optimal* and *prior* proposals respectively.

3.4.2 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 3.4.1, except that we fix the number of islands, $I = 5$. We vary the number of particles for SMC[4] 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 3.6, 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 3.6 are recorded by running the I islands in a serial fashion. As discussed in Section 3.4.1.3, one could parallelize the computation leading to an additional speedup by a factor of I.

[4]We fix $I = 5$ so that the minimum value of C ($= 100$) corresponds to $C/I = 20$ particles per island. Further improvements could be obtained by 'adapting' I to C as discussed in Section 3.4.1.3.

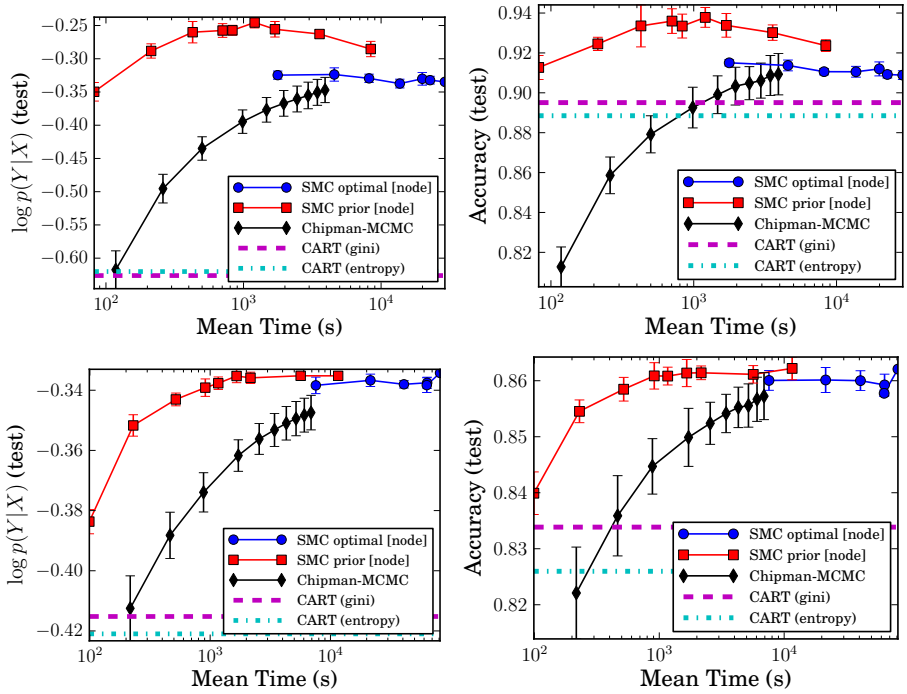


Figure 3.6: Results on *pen-digits* (top row), and *magic-04* (bottom row). Left column plots test $\log p(y|\boldsymbol{x})$ vs runtime, while right column plots test accuracy vs runtime. The blue cirlces, red squares and black diamonds represent *optimal*, *prior* proposals and MCMC respectively.

In the *pen-digits* dataset, the performance of *prior* proposal seems to drop as we increase C beyond 2000. However, we observed that the marginal likelihood on the training data increases with C. The log marginal likelihood of the training data for different proposals is shown in Figure 3.7. As the number of particles increases, the log marginal likelihood of *prior* and *optimal* proposals converge to the same value (as expected). 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).*

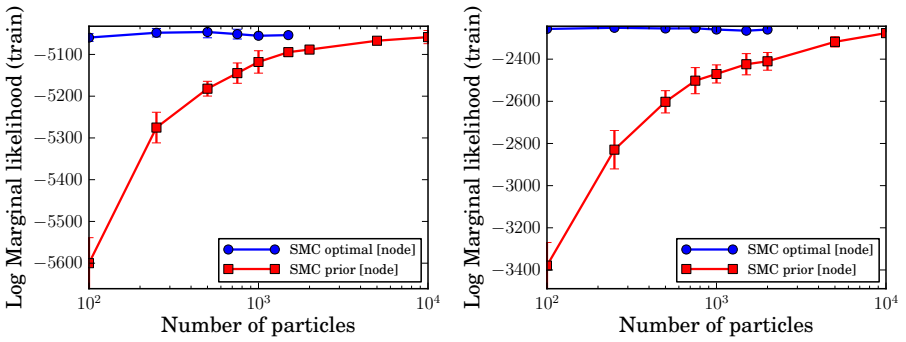


Figure 3.7: Results on *pen-digits* (left), and *magic-04* (right). Mean log marginal likelihood (i.e., mean $\log p(Y|\boldsymbol{X})$ for training data averaged across 10 runs) vs number of particles. The blue circles and red squares represent *optimal* and *prior* proposals respectively.

3.4.3 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 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 3.8, 3.9, 3.10 and 3.11 display the results on *pen-digits* (top row), and *magic-04* (bottom row). The left column plots test $\log p(y|\boldsymbol{x})$ 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 3.6, 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 3.4.2: (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).

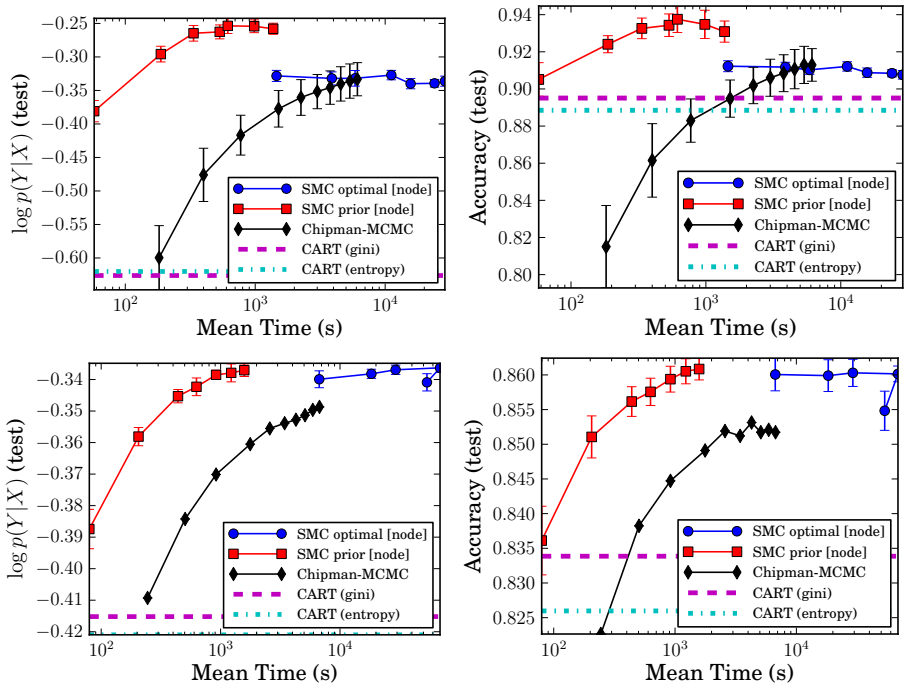


Figure 3.8: Results for the following hyperparameters: $\alpha = 5.0, \boldsymbol{\alpha_s = 0.8}, \beta_s = 0.5$ (see main text for additional information).

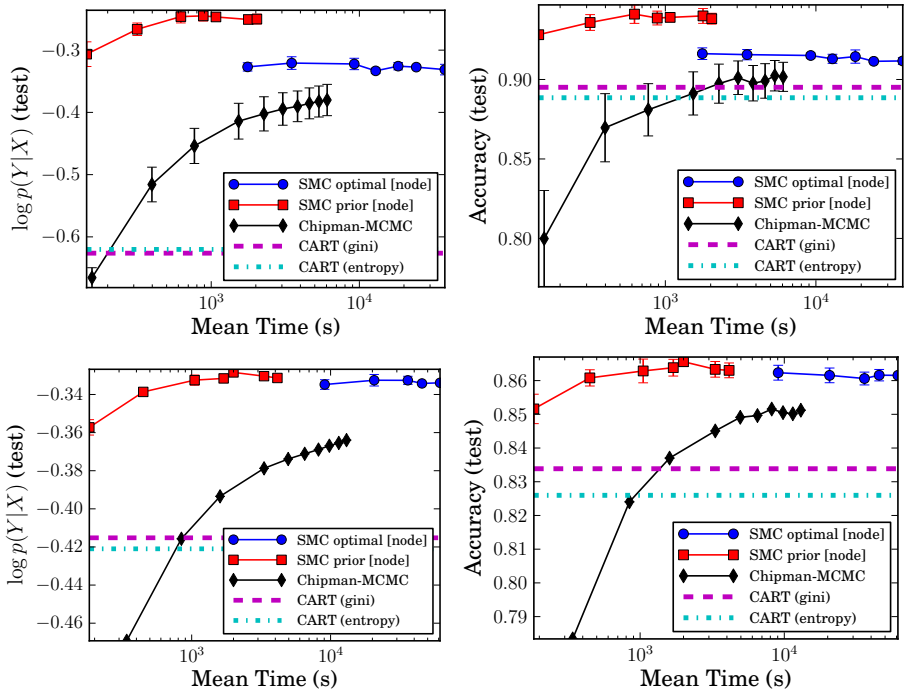


Figure 3.9: Results for the following hyperparameters: $\alpha = 5.0, \alpha_s = 0.95, \boldsymbol{\beta_s = 0.2}$ (see main text for additional information).

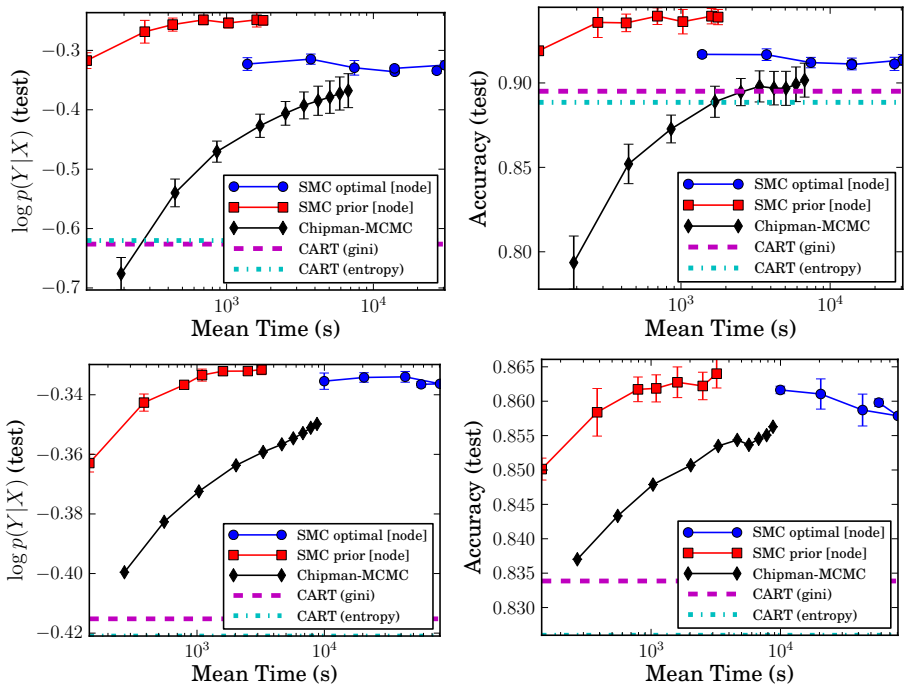


Figure 3.10: Results for the following hyperparameters: $\alpha = 5.0, \boldsymbol{\alpha_s = 0.8, \beta_s = 0.2}$ (see main text for additional information).

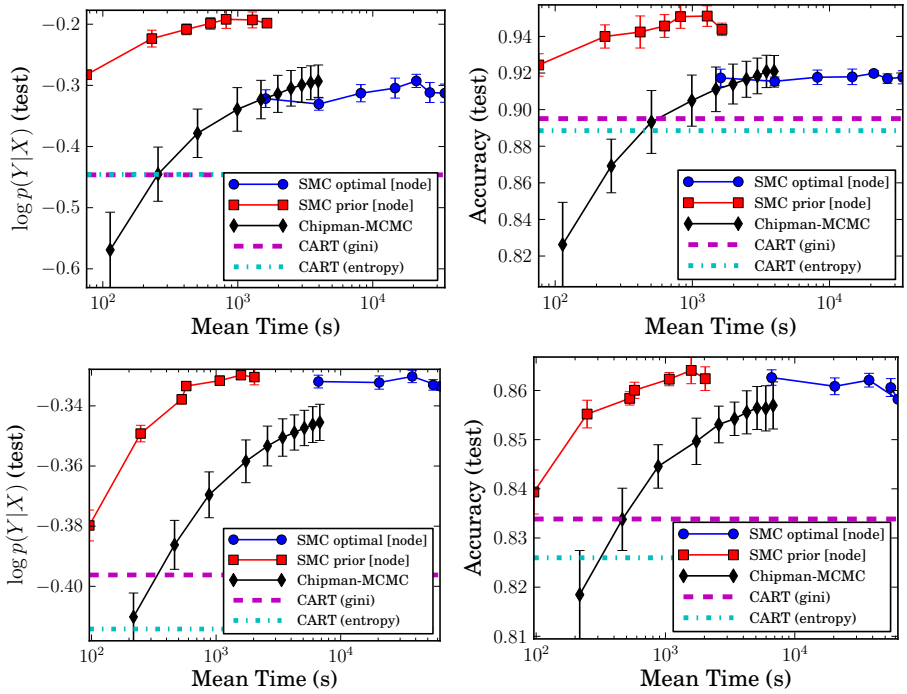


Figure 3.11: Results for the following hyperparameters: $\boldsymbol{\alpha = 1.0}, \alpha_s = 0.95, \beta_s = 0.5$ (see main text for additional information).

3.4.4 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 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).[5] 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 $50\times$ to $100\times$ 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 3.6. 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

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

the benefits of having an approximation to the posterior, but in a fraction of the time required by existing MCMC methods.

3.5 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; for instance, one can choose a subset of input dimensions at random like the prior proposal, then incorporate the local likelihoods for these dimensions like the optimal proposal. Such an algorithm has a similar flavor as the bagging framework exemplified by random forests. We have focused on posterior inference for a fixed set of hyperparameters in this work. However, the SMC algorithm provides an estimate of the marginal likelihood which can be used for learning the hyperparameters, e.g. via Bayesian model selection.

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; we discuss one such extension in Chapter 4.

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 $\boldsymbol{X}$. 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 and Teh, 2009), whose projectivity would re-establish exchangeability while allowing for efficient posterior *computations* that depend on data. Another interesting direction would be to incorporate structured priors for the node parameters as opposed to the independent prior for the leaf node parameters. It would be interesting to extend the proposed SMC algorithm to decision trees with hierarchical priors for classification (e.g. the hierarchy of normalized stable processes discussed in Chapter 5) and regression (e.g. the prior proposed by Chipman and McCulloch (2000) or the hierarchical Gaussian prior discussed in Chapter 6).

Chapter 4

Particle Gibbs for Bayesian additive regression trees

4.1 Introduction

Ensembles of regression trees are at the heart of many state-of-the-art approaches for nonparametric regression (Caruana and Niculescu-Mizil, 2006), and can be broadly classified into two families: *randomized independent regression trees*, wherein the trees are grown independently and predictions are averaged to reduce variance, and *additive regression trees*, wherein each tree fits the residual not explained by the remainder of the trees. In the former category are bagged decision trees (Breiman, 1996), random forests (Breiman, 2001), extremely randomized trees (Geurts et al., 2006), and many others, while additive regression trees can be further categorized into those that are fit in a serial fashion, like gradient boosted regression trees (Friedman, 2001), and those fit in an iterative fashion, like Bayesian additive regression trees (BART) (Chipman et al., 2010) and additive groves (Sorokina et al., 2007).

Among additive approaches, BART is extremely popular and has been successfully applied to a wide variety of problems including protein-DNA binding, credit risk modeling, automatic phishing/spam detection, and drug discovery (Chipman et al., 2010). Additive regression trees must be regularized to avoid overfitting (Friedman, 2002): in BART, over-fitting is controlled by a prior distribution preferring simpler tree structures and non-extreme predictions at leaves. The posterior distribution underlying BART delivers a variety of inferential quantities beyond predictions, including credible intervals for those predictions as well as a measure of variable importance. At the same time, BART has been shown to achieve predictive performance comparable to random forests, boosted regression trees, support vector machines, and neural networks (Chipman et al., 2010).

The standard inference algorithm for BART is an iterative Bayesian backfitting Markov

Chain Monte Carlo (MCMC) algorithm (Hastie et al., 2000). In particular, the MCMC algorithm introduced by Chipman et al. (2010) proposes local changes to individual trees. This sampler can be computationally expensive for large datasets, and so recent work on scaling BART to large datasets (Pratola et al., 2013) considers using only a subset of the moves proposed by Chipman et al. (2010). However, this smaller collection of moves has been observed to lead to poor mixing (Pratola, 2013) which in turn produces an inaccurate approximation to the posterior distribution. While a poorly mixing Markov chain might produce a reasonable prediction in terms of mean squared error, BART is often used in scenarios where its users rely on posterior quantities, and so there is a need for *computationally efficient samplers that mix well* across a range of hyper-parameter settings.

In this work, we describe a novel sampler for BART based on (1) the Particle Gibbs (PG) framework proposed by Andrieu et al. (2010) and (2) the top-down sequential Monte Carlo algorithm for Bayesian decision trees proposed in Chapter 3. Loosely speaking, PG is the *particle version of the Gibbs sampler* where proposals from the exact conditional distributions are replaced by conditional versions of a sequential Monte Carlo (SMC) algorithm. The complete sampler follows the Bayesian backfitting MCMC framework for BART proposed by Chipman et al. (2010); the key difference is that trees are sampled using PG instead of the local proposals used by Chipman et al. (2010). Our sampler, which we refer to as *PG-BART*, approximately samples complete trees from the conditional distribution over a tree fitting the residual. As the experiments bear out, the PG-BART sampler explores the posterior distribution more efficiently than samplers based on local moves. Of course, one could easily consider non-local moves in a Metropolis–Hastings (MH) scheme by proposing complete trees from the tree prior, however these moves would be rejected, leading to slow mixing, in high-dimensional and large data settings. The PG-BART sampler succeeds not only because non-local moves are considered, but because those non-local moves have high posterior probability. Another advantage of the PG sampler is that it only requires one to be able to sample from the prior and does not require evaluation of tree prior in the acceptance ratio unlike (local) MH[1]—hence PG can be computationally efficient in situations where the tree prior is expensive (or impossible) to compute, but relatively easier to sample from.

The chapter is organized as follows: in Section 4.2, we review the BART model; in Section 4.3, we review the MCMC framework proposed by Chipman et al. (2010) and describe the PG sampler in detail. In Section 4.4, we present experiments that compare the PG sampler to existing samplers for BART.

[1]The tree prior term cancels out in the MH acceptance ratio if complete trees are sampled. However, sampling complete trees from the tree prior would lead to very low acceptance rates as discussed earlier.

4.2 Model and notation

In this section, we briefly review decision trees and the BART model. We refer the reader to the paper of Chipman et al. (2010) for further details about the model. Our notation closely follows their's.

4.2.1 Problem setup

We assume that the training data consist of N i.i.d. samples $\boldsymbol{X} = \{\boldsymbol{x}_n\}_{n=1}^N$, where $\boldsymbol{x}_n \in \mathbb{R}^D$, along with corresponding labels $\boldsymbol{Y} = \{y_n\}_{n=1}^N$, where $y_n \in \mathbb{R}$. We focus only on the regression task in this chapter, although the PG sampler can also be used for classification by combining our ideas with the work of Chipman et al. (2010) and Zhang and Härdle (2010).

4.2.2 Regression trees

We refer to Section 2.2 and Figure 2.1 for a review of decision trees and our notation. A decision tree used for regression is referred to as a *regression tree*. In a regression tree, each leaf node $j \in \mathsf{leaves}(\mathsf{T})$ is associated with a real-valued parameter $\mu_j \in \mathbb{R}$. Let $\boldsymbol{\mu} = \{\mu_j\}_{j \in \mathsf{leaves}(\mathsf{T})}$ denote the collection of all parameters. Given a tree $\mathcal{T}$ and a data point $\boldsymbol{x}$, let $\mathsf{leaf}(\boldsymbol{x})$ be the unique leaf node $j \in \mathsf{leaves}(\mathsf{T})$ such that $\boldsymbol{x} \in B_j$, and let $g(\,\cdot\,; \mathcal{T}, \boldsymbol{\mu})$ be the response function associated with $\mathcal{T}$ and $\boldsymbol{\mu}$, given by

$$g(\boldsymbol{x}; \mathcal{T}, \boldsymbol{\mu}) := \mu_{\mathsf{leaf}(\boldsymbol{x})}. \tag{4.1}$$

4.2.3 Likelihood specification for BART

BART is a *sum-of-trees* model, i.e., BART assumes that the label y for an input $\boldsymbol{x}$ is generated by an additive combination of M regression trees. More precisely,

$$y = \sum_{m=1}^{M} g(\boldsymbol{x}; \mathcal{T}_m, \boldsymbol{\mu}_m) + e, \tag{4.2}$$

where $e \sim \mathcal{N}(0, \sigma^2)$ is an independent Gaussian noise term with zero mean and variance σ^2. Hence, the likelihood for a training instance is

$$\ell(y | \{\mathcal{T}_m, \boldsymbol{\mu}_m\}_{m=1}^M, \sigma^2, \boldsymbol{x}) = \mathcal{N}\big(y | \sum_{m=1}^{M} g(\boldsymbol{x}; \mathcal{T}_m, \boldsymbol{\mu}_m), \sigma^2\big),$$

and the likelihood for the entire training dataset is

$$\ell(\boldsymbol{Y} | \{\mathcal{T}_m, \boldsymbol{\mu}_m\}_{m=1}^M, \sigma^2, \boldsymbol{X}) = \prod_n \ell(y_n | \{\mathcal{T}_m, \boldsymbol{\mu}_m\}_{m=1}^M, \sigma^2, \boldsymbol{x}_n).$$

4.2.4 Prior specification for BART

The parameters of the BART model are the noise variance σ^2 and the regression trees $(\mathcal{T}_m, \boldsymbol{\mu}_m)$ for $m = 1, \ldots, M$. The conditional independencies in the prior are captured by the factorization

$$p(\{\mathcal{T}_m, \boldsymbol{\mu}_m\}_{m=1}^M, \sigma^2 | \boldsymbol{X}) = p(\sigma^2) \prod_{m=1}^M p(\boldsymbol{\mu}_m | \mathcal{T}_m) p(\mathcal{T}_m | \boldsymbol{X}).$$

The prior over decision trees $p(\mathcal{T}_m = \{\mathsf{T}_m, \boldsymbol{\delta}_m, \boldsymbol{\xi}_m\} | \boldsymbol{X})$ can be described by the following generative process (Chipman et al., 2010; Lakshminarayanan et al., 2013): Starting with a tree comprised only of a root node ϵ, the tree is grown by deciding once for every node j whether to 1) *stop* and make j a leaf, or 2) *split*, making j an internal node, and add $j0$ and $j1$ as children. The same stop/split decision is made for the children, and their children, and so on. Let ρ_j be a binary indicator variable for the event that j is split. Then every node j is split independently with probability

$$p(\rho_j = 1) = \frac{\alpha_s}{(1 + \mathsf{depth}(j))^{\beta_s}} \mathbb{1}[\text{valid split exists below } j \text{ in } \boldsymbol{X}], \tag{4.3}$$

where the indicator $\mathbb{1}[...]$ forces the probability to be zero when every possible split of j is invalid, i.e., one of the children nodes contains no training data.[2] Informally, the hyperparameters $\alpha_s \in (0, 1)$ and $\beta_s \in [0, \infty)$ control the depth and number of nodes in the tree. Higher values of α_s lead to deeper trees while higher values of β_s lead to shallower trees.

In the event that a node j is split, the dimension δ_j and location ξ_j of the split are assumed to be drawn independently from a uniform distribution over the set of all valid splits of j. The decision tree prior is thus

$$p(\mathcal{T} | \boldsymbol{X}) = \prod_{j \in \mathsf{T} \setminus \mathsf{leaves}(\mathsf{T})} p(\rho_j = 1) \mathcal{U}(\delta_j) \mathcal{U}(\xi_j | \delta_j) \prod_{j \in \mathsf{leaves}(\mathsf{T})} p(\rho_j = 0), \tag{4.4}$$

where $\mathcal{U}(\cdot)$ denotes the probability mass function of the uniform distribution over dimensions that contain at least one valid split, and $\mathcal{U}(\cdot | \delta_j)$ denotes the probability density function of the uniform distribution over valid split locations along dimension δ_j in block B_j.

Given a decision tree $\mathcal{T}$, the parameters associated with its leaves are independent and identically distributed normal random variables, and so

$$p(\boldsymbol{\mu} | \mathcal{T}) = \prod_{j \in \mathsf{leaves}(\mathsf{T})} \mathcal{N}(\mu_j | m_\mu, \sigma_\mu^2). \tag{4.5}$$

[2] Note that $p(\rho_j = 1)$ depends on $\boldsymbol{X}$ and the split dimensions and locations at the ancestors of j in $\mathcal{T}$ due to the indicator function for valid splits. We elide this dependence to keep the notation simple.

The mean m_μ and variance σ_μ^2 hyperparameters are set indirectly: Chipman et al. (2010) shift and rescale the labels Y such that $y_{\mathsf{min}} = -0.5$ and $y_{\mathsf{max}} = 0.5$, and set $m_\mu = 0$ and $\sigma_\mu = 0.5/k\sqrt{M}$, where $k > 0$ is an hyperparameter. This adjustment has the effect of keeping individual node parameters μ_j small; the higher the values of k and M, the greater the shrinkage towards the mean m_μ.

The prior $p(\sigma^2)$ over the noise variance is an inverse gamma distribution. The hyperparameters ν and q indirectly control the shape and rate of the inverse gamma prior over σ^2. Chipman et al. (2010) compute an overestimate of the noise variance $\widehat{\sigma}^2$, e.g., using the least-squares variance or the unconditional variance of Y, and, for a given shape parameter ν, set the rate such that $\mathbb{P}(\sigma \leq \widehat{\sigma}) = q$, i.e., the qth quantile of the prior over σ is located at $\widehat{\sigma}$.

Chipman et al. (2010) recommend the default values: $\nu = 3, q = 0.9, k = 2, M = 200$ and $\alpha_s = 0.95, \beta_s = 2.0$. Unless otherwise specified, we use this default hyperparameter setting in our experiments.

In Section 3.2.3, we presented a sequential generative process for the tree prior $p(\mathcal{T}|\boldsymbol{X})$, where a tree $\mathcal{T}$ is generated by starting from an empty tree $\mathcal{T}_{(0)}$ and sampling a sequence $\mathcal{T}_{(1)}, \mathcal{T}_{(2)}, \ldots$ of partial trees.[3] We will leverage this sequential representation for our PG sampler. We refer to Section 3.2.3 for the details and Figure 3.1 for a cartoon of the sequential generative process. In Section 3.2.3, we discussed a more general version where more than one node may be expanded in an iteration. Based on the experimental results comparing different expansion strategies in Section 3.4.1, we restrict our attention here to *node-wise expansion*: one node is expanded per iteration and the nodes are expanded in a breadth-wise fashion.

Algorithm 4.1 Bayesian backfitting MCMC for posterior inference in BART

Inputs: Training data $(\boldsymbol{X}, Y)$, BART hyperparameters $(\nu, q, k, M, \alpha_s, \beta_s)$
Initialization: For all m, set $\mathcal{T}_m^{(0)} = \{\mathsf{T}_m^{(0)} = \{\epsilon\}, \boldsymbol{\xi}_m^{(0)} = \boldsymbol{\delta}_m^{(0)} = \emptyset\}$ and sample $\boldsymbol{\mu}_m^{(0)}$
for $i = 1 : \mathsf{max\_iter}$ **do**
    Sample $\sigma^{2(i)}|\mathcal{T}_{1:M}^{(i-1)}, \boldsymbol{\mu}_{1:M}^{(i-1)}$ ▷ *sample from inverse gamma distribution*
    for $m = 1 : M$ **do**
        Compute residual $R_m^{(i)}$ ▷ *using (4.7)*
        Sample $\mathcal{T}_m^{(i)}|R_m^{(i)}, \sigma^{2(i)}, \mathcal{T}_m^{(i-1)}$ ▷ *using CGM, GrowPrune or PG*
        Sample $\boldsymbol{\mu}_m^{(i)}|R_m^{(i)}, \sigma^{2(i)}, \mathcal{T}_m^{(i)}$ ▷ *sample from Gaussian distribution*

4.3 Posterior inference for BART

In this section, we briefly review the MCMC framework proposed in (Chipman et al., 2010), discuss limitations of existing samplers and then present our PG sampler.

[3] Note that $\mathcal{T}_{(t)}$ denotes partial tree at stage t, whereas $\mathcal{T}_m$ denotes the mth tree in the ensemble.

4.3.1 MCMC for BART

Given the likelihood and the prior, our goal is to compute the posterior distribution

$$p(\{\mathcal{T}_m, \boldsymbol{\mu}_m\}_{m=1}^M, \sigma^2 | Y, \boldsymbol{X}) \propto \ell(Y | \{\mathcal{T}_m, \boldsymbol{\mu}_m\}_{m=1}^M, \sigma^2, \boldsymbol{X})\, p(\{\mathcal{T}_m, \boldsymbol{\mu}_m\}_{m=1}^M, \sigma^2 | \boldsymbol{X}). \quad (4.6)$$

Chipman et al. (2010) proposed a Bayesian backfitting MCMC to sample from the BART posterior. At a high level, the Bayesian backfitting MCMC is a Gibbs sampler that loops through the trees, sampling each tree $\mathcal{T}_m$ and associated parameters $\boldsymbol{\mu}_m$ conditioned on σ^2 and the remaining trees and their associated parameters $\{\mathcal{T}_{m'}, \boldsymbol{\mu}_{m'}\}_{m' \neq m}$, and samples σ^2 conditioned on all the trees and parameters $\{\mathcal{T}_m, \boldsymbol{\mu}_m\}_{m=1}^M$. Let $\mathcal{T}_m^{(i)}, \boldsymbol{\mu}_m^{(i)}$, and $\sigma^{2(i)}$ respectively denote the values of $\mathcal{T}_m, \boldsymbol{\mu}_m$ and σ^2 at the m^{th} MCMC iteration. Sampling σ^2 conditioned on $\{\mathcal{T}_m, \boldsymbol{\mu}_m\}_{m=1}^M$ is straightforward due to conjugacy. To sample $\mathcal{T}_m, \boldsymbol{\mu}_m$ conditioned on the other trees $\{\mathcal{T}_{m'}, \boldsymbol{\mu}_{m'}\}_{m' \neq m}$, we first sample $\mathcal{T}_m | \{\mathcal{T}_{m'}, \boldsymbol{\mu}_{m'}\}_{m' \neq m}, \sigma^2$ and then sample $\boldsymbol{\mu}_m | \mathcal{T}_m, \{\mathcal{T}_{m'}, \boldsymbol{\mu}_{m'}\}_{m' \neq m}, \sigma^2$. (Note that $\boldsymbol{\mu}_m$ is integrated out while sampling $\mathcal{T}_m$.) More precisely, we compute the residual

$$R_m = Y - \textstyle\sum_{m'=1, m' \neq m}^M g(\boldsymbol{X}; \mathcal{T}_{m'}, \boldsymbol{\mu}_{m'}). \quad (4.7)$$

Using the residual $R_m^{(i)}$ as the target, Chipman et al. (2010) sample $\mathcal{T}_m^{(i)}$ by proposing local changes to $\mathcal{T}_m^{(i-1)}$. Finally, $\boldsymbol{\mu}_m$ is sampled from a Gaussian distribution conditioned on $\mathcal{T}_m, \{\mathcal{T}_{m'}, \boldsymbol{\mu}_{m'}\}_{m' \neq m}, \sigma^2$. The procedure is summarized in Algorithm 4.1.

4.3.2 Existing samplers for BART

To sample $\mathcal{T}_m$, Chipman et al. (2010) use the MCMC algorithm proposed by Chipman et al. (1998). This algorithm, which we refer to as CGM, is a *Metropolis-within-Gibbs* sampler that randomly chooses one of the following four moves: *grow* (which randomly chooses a leaf node and splits it further into left and right children), *prune* (which randomly chooses an internal node where both the children are leaf nodes and prunes the two leaf nodes, thereby making the internal node a leaf node), *change* (which changes the decision rule at a randomly chosen internal node), *swap* (which swaps the decision rules at a parent-child pair where both the parent and child are internal nodes). There are two issues with the CGM sampler: (1) the CGM sampler makes local changes to the tree, which is known to affect mixing when computing the posterior over a single decision tree (Wu et al., 2007). Chipman et al. (2010) claim that the default hyper-parameter values encourage shallower trees and hence mixing is not affected significantly. However, if one wishes to use BART on large datasets where individual trees are likely to be deeper, the CGM sampler might suffer from mixing issues. (2) The *change* and *swap* moves in CGM sampler are computationally expensive for large datasets that involve deep trees (since they involve re-computation of all likelihoods in the subtree below the top-most node affected by the proposal). For computational

efficiency, Pratola et al. (2013) propose using only the *grow* and *prune* moves; we will call this the GrowPrune sampler. However, as we illustrate in Section 4.4, the GrowPrune sampler can inefficiently explore the posterior in scenarios where there are multiple possible trees that explain the observations equally well. In the next section, we present a novel sampler that addresses both of these concerns.

4.3.3 PG sampler for BART

Recall that Chipman et al. (2010) sample $\mathcal{T}_m^{(i)}$ using $R_m^{(i)}$ as the target by proposing local changes to $\mathcal{T}_m^{(i-1)}$. It is natural to ask if it is possible to sample a complete tree $\mathcal{T}_m^{(i)}$ rather than just local changes. Indeed, this is possible by marrying the sequential representation of the tree proposed in Chapter 3 with the Particle Markov Chain Monte Carlo (PMCMC) framework (Andrieu et al., 2010) where an SMC algorithm (particle filter) is used as a high-dimensional proposal for MCMC. The PG sampler is implemented using the so-called *conditional SMC* algorithm (instead of the Metropolis-Hastings samplers described in Section 4.3.2) in line 7 of Algorithm 4.1. At a high level, the conditional SMC algorithm is similar to the SMC algorithm proposed in Chapter 3, except that one of the particles is clamped to the current tree $\mathcal{T}_m^{(i-1)}$.

Before describing the PG sampler, we derive the conditional posterior $\mathcal{T}_m|\{\mathcal{T}_{m'}, \boldsymbol{\mu}_{m'}\}_{m'\neq m}, \sigma^2, Y, \boldsymbol{X}$. Let N_j denote the set of data point indices $n \in \{1, \ldots, N\}$ such that $\boldsymbol{x}_n \in B_j$. Slightly abusing the notation, let R_{N_j} denote the vector containing residuals of data points in node j. Given $R := Y - \sum_{m'\neq m} g(\boldsymbol{X}; \mathcal{T}_{m'}, \boldsymbol{\mu}_{m'})$, it is easy to see that the conditional posterior over $\mathcal{T}_m, \boldsymbol{\mu}_m$ is given by

$$p(\mathcal{T}_m, \boldsymbol{\mu}_m|\{\mathcal{T}_{m'}, \boldsymbol{\mu}_{m'}\}_{m'\neq m}, \sigma^2, Y, \boldsymbol{X}) \propto p(\mathcal{T}_m|\boldsymbol{X}) \prod_{j\in \mathsf{leaves}(\mathsf{T}_m)} \prod_{n\in N_j} \mathcal{N}(R_n|\mu_j, \sigma^2)\mathcal{N}(\mu_j|m_\mu, \sigma_\mu^2).$$

Let $\pi(\mathcal{T}_m)$ denote the conditional posterior over $\mathcal{T}_m$. Integrating out $\boldsymbol{\mu}$ and using (4.4) for $p(\mathcal{T}_m|\boldsymbol{X})$, the conditional posterior $\pi(\mathcal{T}_m)$ is

$$\pi(\mathcal{T}_m) = p(\mathcal{T}_m|\{\mathcal{T}_{m'}, \boldsymbol{\mu}_{m'}\}_{m'\neq m}, \sigma^2, Y, \boldsymbol{X}) \propto p(\mathcal{T}_m|\boldsymbol{X}) \prod_{j\in \mathsf{leaves}(\mathsf{T}_m)} p(R_{N_j}|\sigma^2, m_\mu, \sigma_\mu^2), \tag{4.8}$$

where $p(R_{N_j}|\sigma^2, m_\mu, \sigma_\mu^2)$ denotes the marginal likelihood at a node j, given by

$$p(R_{N_j}|\sigma^2, m_\mu, \sigma_\mu^2) = \int_{\mu_j} \prod_{n\in N_j} \mathcal{N}(R_n|\mu_j, \sigma^2)\mathcal{N}(\mu_j|m_\mu, \sigma_\mu^2)d\mu_j. \tag{4.9}$$

The goal is to sample from the (conditional) posterior distribution $\pi(\mathcal{T}_m)$. In Chapter 3, we presented a top-down particle filtering algorithm that approximates the posterior over decision trees. Since this SMC algorithm can sample complete trees, it is tempting

to substitute an exact sample from $\pi(\mathcal{T}_m)$ with an approximate sample from the particle filter. However, Andrieu et al. (2010) observed that this naive approximation does not leave the joint posterior distribution (4.6) invariant, and so they proposed instead to generate a sample using a modified version of the SMC algorithm, which they called the *conditional-SMC algorithm*, and demonstrated that this leaves the joint distribution (4.6) invariant. (We refer the reader to the paper by Andrieu et al. (2010) for further details about the PMCMC framework.) By building off the top-down particle filter for decision trees, we can define a conditional-SMC algorithm for sampling from $\pi(\mathcal{T}_m)$.

The conditional-SMC algorithm is an MH kernel with $\pi(\mathcal{T}_m)$ as its stationary distribution. To reduce clutter, let $\mathcal{T}^*$ denote the old tree and $\mathcal{T}$ denote the tree we wish to sample. The conditional-SMC algorithm samples $\mathcal{T}$ from a C-particle approximation of $\pi(\mathcal{T})$, which can be written as $\sum_{c=1}^{C} \overline{w}(c)\delta_{\mathcal{T}(c)}$ where $\mathcal{T}(c)$ denotes the c^{th} tree (particle) and the weights sum to 1, that is, $\sum_c \overline{w}(c) = 1$.

SMC proposal: Each particle $\mathcal{T}(c)$ is the end product of a sequence of partial trees $\mathcal{T}_{(0)}(c), \mathcal{T}_{(1)}(c), \mathcal{T}_{(2)}(c), \ldots$, and the weight $\overline{w}(c)$ reflects how well the c^{th} tree explains the residual R. One of the particles, say the first particle, without loss of generality, is clamped to the old tree $\mathcal{T}^*$ at all stages of the particle filter, i.e., $\mathcal{T}_{(t)}(1) = \mathcal{T}^*_{(t)}$. At stage t, the remaining $C-1$ particles are sampled from the sequential generative process $\mathbb{P}_t(\cdot \,|\, \mathcal{T}_{(t-1)}(c))$ described in Section 3.2.3. Unlike state space models where the length of the latent state sequence is fixed, the sampled decision tree sequences may be of different length and could potentially be deeper than the old tree $\mathcal{T}^*$. Hence, whenever $E_{(t)} = \emptyset$, we set $\mathbb{P}_{(t)}(\mathcal{T}_{(t)}|\mathcal{T}_{(t-1)}) = \delta_{\mathcal{T}_{(t-1)}}$, i.e., $\mathcal{T}_{(t)} = \mathcal{T}_{(t-1)}$.

SMC weight update: Since the prior is used as the proposal, the particle weight $w_{(t)}(c)$ is multiplicatively updated with the ratio of the marginal likelihood of $\mathcal{T}_{(t)}(c)$ to the marginal likelihood of $\mathcal{T}_{(t-1)}(c)$. The marginal likelihood associated with a (partial) tree $\mathcal{T}$ is a product of the marginal likelihoods associated with the leaf nodes of $\mathcal{T}$ defined in (4.9). As in Chapter 3, we treat the eligible nodes $E_{(t)}$ as leaf nodes while computing the marginal likelihood for a partial tree $\mathcal{T}_{(t)}$. Plugging in (4.9), the SMC weight update is given by (4.10) in Algorithm 4.2.

Resampling: The resampling step in the conditional-SMC algorithm is slightly different from the typical SMC resampling step. Recall that the first particle is always clamped to the old tree. The remaining $C-1$ particles are resampled such that the probability of choosing particle c is proportional to its weight $w_{(t)}(c)$. We used multinomial resampling in our experiments, although other resampling strategies are possible.

When none of the trees contain eligible nodes, the conditional-SMC algorithm stops and returns a sample from the particle approximation. Without loss of generality, we assume that the C^{th} particle is returned. The PG sampler is summarized in Algorithm 4.2.

The computational complexity of the conditional-SMC algorithm in Algorithm 4.2 is similar to that of the top-down particle filtering algorithm in Section 3.3.2. Even

though the PG sampler has a higher per-iteration complexity in general compared to GrowPrune and CGM samplers, it can mix faster since it can propose a completely different tree that explains the data. The GrowPrune sampler requires many iterations to explore multiple modes (since a prune operation is likely to be rejected around a mode). The CGM sampler can change the decisions at internal nodes; however, it is inefficient since a change in an internal node that leaves any of the nodes in the subtree below empty will be rejected. We demonstrate the competitive performance of PG in the experimental section.

Algorithm 4.2 Conditional-SMC algorithm used in the PG-BART sampler

Inputs: Training data: features $\boldsymbol{X}$, 'target' R ▷ *R denotes residual in BART*
  Number of particles C
  Old tree $\mathcal{T}^*$ (along with the partial tree sequence $\mathcal{T}^*_{(0)}, \mathcal{T}^*_{(1)}, \mathcal{T}^*_{(2)}, \dots$)
Initialize: $(\forall c)$, set $\mathsf{T}_{(0)}(c) = E_{(0)}(c) = \{\epsilon\}$ and $\boldsymbol{\xi}_{(0)}(c) = \boldsymbol{\delta}_{(0)}(c) = \emptyset$
  $(\forall c)$, set weights $w_{(0)}(c) = p(R_{N_\epsilon}|\sigma^2, m_\mu, \sigma^2_\mu)$ and $W_{(0)} = \sum_c w_{(0)}(c)$
for $t = 1 :$ max-stages **do**
  Set $\mathcal{T}_{(t)}(1) = \mathcal{T}^*_{(t)}$ ▷ *clamp the first particle to the partial tree of $\mathcal{T}^*$ at stage t*
  for $c = 2 : C$ **do**
    Sample $\mathcal{T}_{(t)}(c)$ from $\mathbb{P}_{(t)}(\cdot \,|\, \mathcal{T}_{(t-1)}(c))$ where
    $\mathcal{T}_{(t)}(c) := (\mathsf{T}_{(t)}(c), \boldsymbol{\delta}_{(t)}(c), \boldsymbol{\xi}_{(t)}(c), E_{(t)}(c))$ ▷ *section 3.2.3*
  for $c = 1 : C$ **do**
    ▷ *If $E_{(t-1)}(c)$ is non-empty, let j denote the node popped from $E_{(t-1)}(c)$.*
    Update weights:

$$w_{(t)}(c) = \begin{cases} w_{(t-1)}(c) & \text{if } E_{(t-1)}(c) \text{ is empty or } j \text{ is stopped,} \\ w_{(t-1)}(c) \dfrac{\prod_{j'=j0,j1} p(R_{N_{j'}}|\sigma^2, m_\mu, \sigma^2_\mu)}{p(R_{N_j}|\sigma^2, m_\mu, \sigma^2_\mu)} & \text{if } j \text{ is split.} \end{cases} \tag{4.10}$$

  Compute normalization: $W_{(t)} = \sum_c w_{(t)}(c)$
  Normalize weights: $(\forall c)\, \overline{w}_{(t)}(c) = w_{(t)}(c)/W_{(t)}$
  Set $a_1 = 1$ and for $c = 2 : C$, resample indices a_c from $\sum_{c'} \overline{w}_{(t)}(c')\delta_{c'}$ ▷ *resample all particles except the first*
  $(\forall c)\, \mathcal{T}_{(t)}(c) \leftarrow \mathcal{T}_{(t)}(a_c)$; $w_{(t)}(c) \leftarrow W_{(t)}/C$
  if $(\forall c)\, E_{(t)}(c) = \emptyset$ **then** exit for loop
return $\mathcal{T}_{(t)}(C) = (\mathsf{T}_{(t)}(C), \boldsymbol{\delta}_{(t)}(C), \boldsymbol{\xi}_{(t)}(C))$ ▷ *return a sample from the approximation $\sum_{c'} \overline{w}_{(t)}(c')\delta_{\mathcal{T}_{(t)}(c')}$ to line 7 of Algorithm 4.1*

4.4 Experimental evaluation

In this section, we present experimental comparisons between the PG sampler and existing samplers for BART. Since the main contribution of this work is a different inference algorithm for an existing model, we just compare the efficiency of the inference algorithms and do not compare to other models. BART has been shown to demonstrate excellent prediction performance compared to other popular black-box non-linear

regression approaches; we refer the interested reader to Chipman et al. (2010).

We implemented all the samplers in Python and ran experiments on the same desktop machine so that the timing results are comparable. The scripts can be downloaded from the authors' webpages.[4] We set the number of particles $C = 10$ for computational efficiency[5] and max-stages $= 5000$, following Chapter 3, although the algorithm always terminated much earlier.

4.4.1 Hypercube-D dataset

We investigate the performance of the samplers on a dataset where there are multiple trees that explain the residual (conditioned on other trees). This problem is equivalent to posterior inference over a decision tree where the labels are equal to the residual. Hence, we generate a synthetic dataset where multiple trees are consistent with the observed labels. Intuitively, a local sampler can be expected to mix reasonably well when the true posterior consists of shallow trees; however, a local sampler will lead to an inefficient exploration when the posterior consists of deep trees. Since the depth of trees in the true posterior is at the heart of the mixing issue, we create synthetic datasets where the depth of trees in the true posterior can be controlled.

We generate the hypercube-D dataset as follows: for each of the 2^D vertices of $[-1, 1]^D$, we sample 10 data points. The $\boldsymbol{x}$ location of a data point is generated as $\boldsymbol{x} = \boldsymbol{v} + \epsilon$ where $\boldsymbol{v}$ is the vertex location and ϵ is a random offset generated as $\epsilon \sim \mathcal{N}(\mathbf{0}, 0.1^2 I_D)$. Each vertex is associated with a different function value and the function values are generated from $\mathcal{N}(0, 3^2)$. Finally the observed label is generated as $y = f + e$ where f denotes the true function value at the vertex and $e \sim \mathcal{N}(0, 0.01^2)$. Figure 4.1 shows a sample hypercube-2 dataset. As D increases, the number of trees that explains the observations increases.

We fix $M = 1, \alpha_s = 0.95$ and set remaining BART hyperparameters to the default values. Since the true tree has 2^D leaves, we set[6] β_s such that the expected number of leaves is roughly 2^D. We run 2000 iterations of MCMC. Figures 4.2, 4.3 and 4.4 illustrates the posterior trace plots for $D = 2$, $D = 3$ and $D = 4$ respectively. We observe that PG converges much faster to the posterior in terms of number of leaves as well as the test MSE. We observe that GrowPrune sampler tends to overestimate the number of leaves; the low value of train MSE indicates that the GrowPrune sampler is stuck close to a mode and is unable to explore the true posterior. Pratola (2013) has reported similar behavior of GrowPrune sampler on a different dataset as well.

[4] http://www.gatsby.ucl.ac.uk/~balaji/pgbart/

[5] Higher values of C increase the computational complexity significantly, while lower values of C lead to poor mixing. We found $C = 10$ to achieve a good tradeoff in our initial experiments and hence fixed $C = 10$ for all of the experiments.

[6] The values of β_s for $D = 2, 3, 4, 5$ and 7 are $1.0, 0.5, 0.4, 0.3$ and 0.25 respectively.

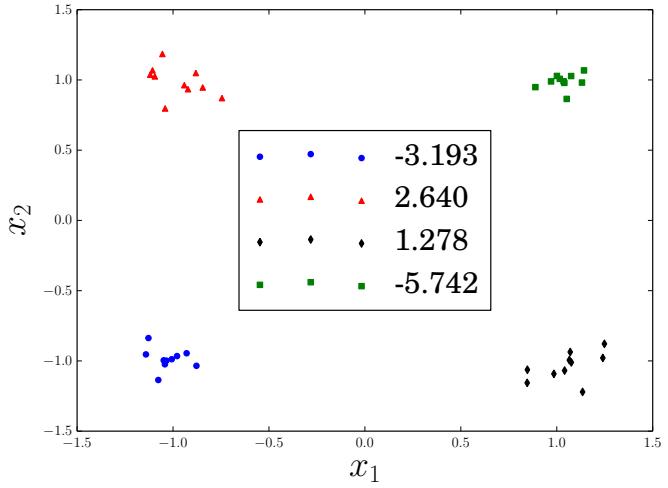


Figure 4.1: Hypercube-2 dataset: see main text for details.

4.4.2 Results on hypercube$-D$ dataset

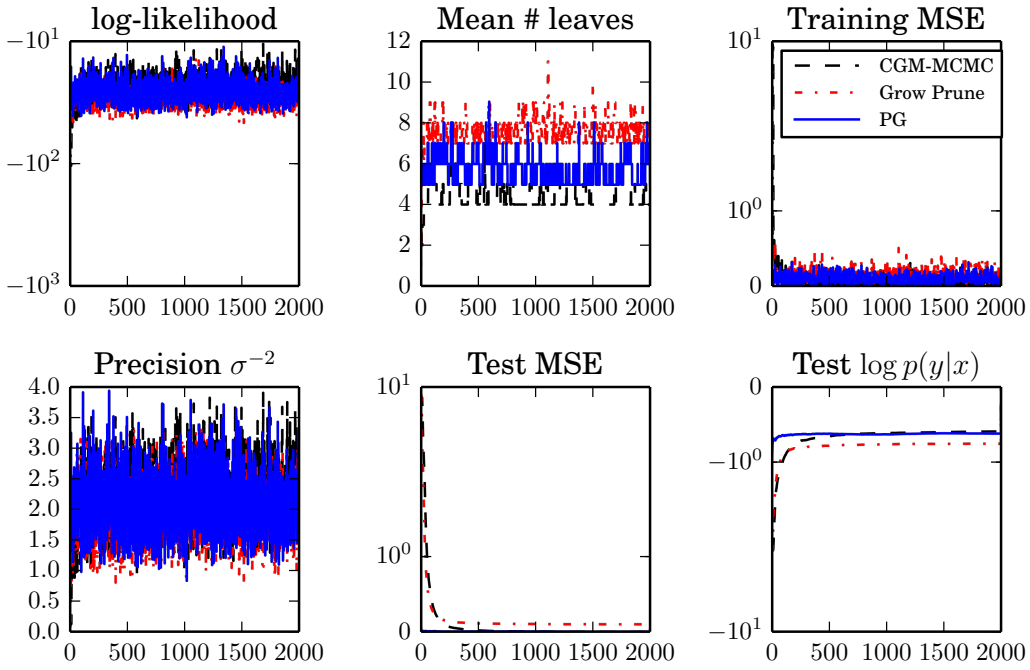


Figure 4.2: Results on Hypercube-2 dataset.

We compare the algorithms by computing effective sample size (ESS). ESS is a measure of how well the chain mixes and is frequently used to assess performance of MCMC algorithms; we compute ESS using R-CODA (Plummer et al., 2006). We discard the first 1000 iterations as burn-in and use the remaining 1000 iterations to compute ESS (on the log-likelihood). Since the per iteration cost of generating a sample differs across samplers, we additionally report ESS per unit time. The ESS (computed using log-likelihood values) and ESS per second (ESS/s) values are shown in Tables 4.1 and

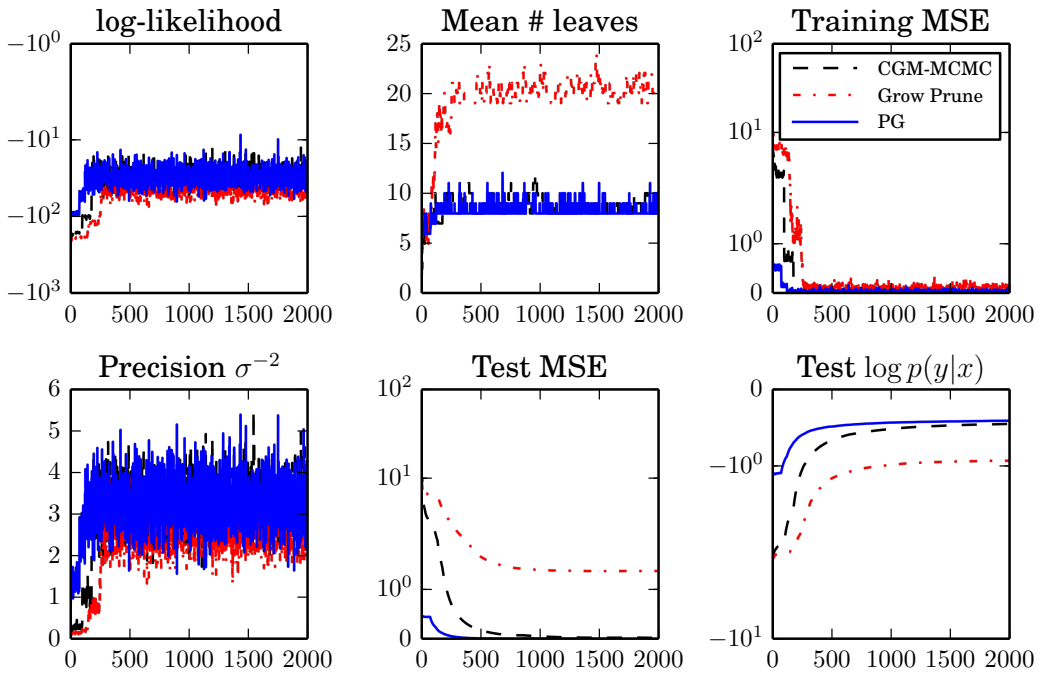


Figure 4.3: Results on Hypercube-3 dataset.

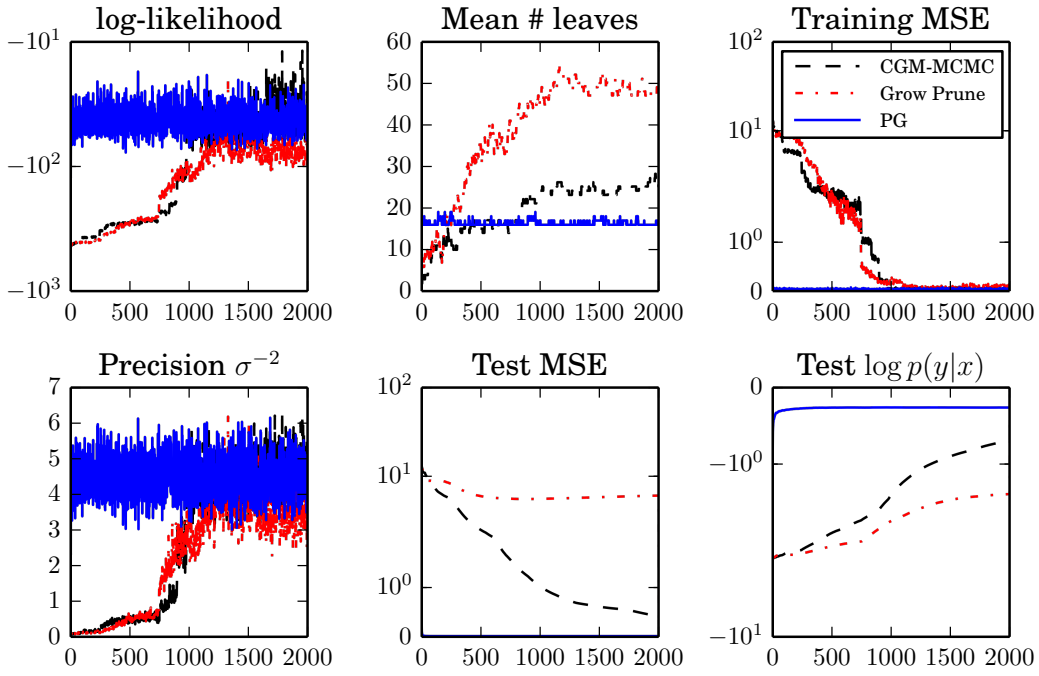


Figure 4.4: Results on Hypercube-4 dataset.

4.2 respectively. When the true tree is shallow ($D = 2$ and $D = 3$), we observe that CGM sampler mixes well and is computationally efficient. However, as the depth of the true tree increases ($D = 4, 5, 7$), PG achieves much higher ESS and ESS/s compared to CGM and GrowPrune samplers.

D	CGM	GrowPrune	PG
2	**751.66**	473.57	259.11
3	**762.96**	285.2	666.71
4	14.01	11.76	**686.79**
5	2.92	1.35	**667.27**
7	1.16	1.78	**422.96**

Table 4.1: Comparison of ESS for CGM, GrowPrune and PG samplers on Hypercube-D dataset.

D	CGM	GrowPrune	PG
2	**157.67**	114.81	7.69
3	**93.01**	26.94	11.025
4	0.961	0.569	**5.394**
5	0.130	0.071	**1.673**
7	0.027	0.039	**0.273**

Table 4.2: Comparison of ESS/s (ESS per second) for CGM, GrowPrune and PG samplers on Hypercube-D dataset.

4.4.3 Real world datasets

In this experiment, we study the effect of the data dimensionality on mixing. Even when the trees are shallow, the number of trees consistent with the labels increases as the data dimensionality increases. Using the default BART prior (which promotes shallower trees), we compare the performance of the samplers on real world datasets of varying dimensionality.

We consider the *CaliforniaHouses*, *YearPredictionMSD* and *CTslices* datasets used by Johnson and Zhang (2013). For each dataset, there are three training sets, each of which contains 2000 data points, and a single test set. The dataset characteristics are summarized in Table 4.3.

Dataset	N_{train}	N_{test}	D
CaliforniaHouses	2000	5000	6
YearPredictionMSD	2000	51630	90
CTslices	2000	24564	384

Table 4.3: Characteristics of datasets.

We run each sampler using the three training datasets and report average ESS and ESS/s. All three samplers achieve very similar MSE to those reported by Johnson and Zhang (2013). The average number of leaves in the posterior trees was found to be small and very similar for all the samplers. Tables 4.4 and 4.5 respectively present results comparing ESS and ESS/s of the different samplers. As the data dimensionality increases, we observe that PG outperforms existing samplers.

Dataset	CGM	GrowPrune	PG
CaliforniaHouses	18.956	34.849	**76.819**
YearPredictionMSD	29.215	21.656	**76.766**
CTslices	2.511	5.025	**11.838**

Table 4.4: Comparison of ESS for CGM, GrowPrune and PG samplers on real world datasets.

Dataset	CGM $\times 10^{-3}$	GrowPrune $\times 10^{-3}$	PG $\times 10^{-3}$
CaliforniaHouses	1.967	**48.799**	16.743
YearPredictionMSD	2.018	7.029	**14.070**
CTslices	0.080	0.615	**2.115**

Table 4.5: Comparison of ESS/s for CGM, GrowPrune and PG samplers on real world datasets.

4.5 Discussion

We have presented a novel PG sampler for BART. Unlike existing samplers which make local moves, PG can propose complete trees. Experimental results confirm that PG dramatically increases mixing when the true posterior consists of deep trees or when the data dimensionality is high. We have shown the benefits of improved mixing in terms of effective sample size; a promising direction would be use the PG sampler in problems such as variable selection (Bleich et al., 2014), where better mixing would lead to more reliable variable importance measures. While we have presented PG only for the BART model, it is applicable to extensions of BART that use a different likelihood model as well. PG can also be used along with other priors for decision trees, e.g., those of Denison et al. (1998), Wu et al. (2007) and Lakshminarayanan et al. (2014). Backward simulation (Lindsten and Schön, 2013) and ancestral sampling (Lindsten et al., 2012) have been shown to significantly improve mixing of PG for state-space models. Extending these ideas to PG-BART is a challenging and interesting future direction.

Chapter 5

Mondrian forests for classification

5.1 Introduction

Despite being introduced over a decade ago by Breiman (2001), random forests remain one of the most popular machine learning tools due in part to their accuracy, scalability, and robustness in real-world classification tasks (Caruana and Niculescu-Mizil, 2006). (We refer to (Criminisi et al., 2012) for an excellent survey of random forests.) In this chapter, we introduce a novel class of random forests—called *Mondrian forests* (MF), due to the fact that the underlying tree structure of each classifier in the ensemble is a so-called *Mondrian process*. Using the properties of Mondrian processes, we present an efficient *online* algorithm that agrees with its batch counterpart at each iteration. Not only are online Mondrian forests faster and more accurate than recent proposals for online random forest methods, but they nearly match the accuracy of state-of-the-art *batch* random forest methods trained on the same dataset.

The chapter is organized as follows: In Section 5.2, we describe our approach at a high-level, and in Sections 5.3, 5.4, and 5.5, we describe the tree structures, label model, and incremental updates/predictions in more detail. We discuss related work in Section 5.6, demonstrate the excellent empirical performance of MF in Section 5.7, and conclude in Section 5.8 with a discussion about future work.

5.2 Approach

Given N labeled examples $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N) \in \mathcal{X} \times \mathcal{Y}$ as training data, our task is to predict labels $y \in \mathcal{Y}$ for unlabeled test points $\boldsymbol{x} \in \mathcal{X}$. We will focus on multi-class classification where $\mathcal{Y} := \{1, \ldots, K\}$, however, it is possible to extend the methodology to other supervised learning tasks such as regression. Let $\boldsymbol{X}_{1:n} := (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, $Y_{1:n} := (y_1, \ldots, y_n)$, and $\mathcal{D}_{1:n} := (\boldsymbol{X}_{1:n}, Y_{1:n})$.

A Mondrian forest classifier is constructed much like a random forest: Given training data $\mathcal{D}_{1:N}$, we sample an independent collection $\mathcal{T}_1, \ldots, \mathcal{T}_M$ of so-called Mondrian trees, which we will describe in the next section. The prediction made by each Mondrian tree $\mathcal{T}_m$ is a distribution $p_{\mathcal{T}_m}(y|\boldsymbol{x}, \mathcal{D}_{1:N})$ over the class label y for a test point $\boldsymbol{x}$. The prediction made by the Mondrian forest is the average $\frac{1}{M}\sum_{m=1}^{M} p_{\mathcal{T}_m}(y|\boldsymbol{x}, \mathcal{D}_{1:N})$ of the individual tree predictions. As $M \to \infty$, the average converges at the standard rate to the expectation $\mathbb{E}_{\mathcal{T}\sim\mathrm{MT}(\lambda,\mathcal{D}_{1:N})}[p_{\mathcal{T}}(y|\boldsymbol{x}, \mathcal{D}_{1:N})]$, where $\mathrm{MT}(\lambda, \mathcal{D}_{1:N})$ is the distribution of a Mondrian tree. As the limiting expectation does not depend on M, we would not expect to see overfitting behavior as M increases. A similar observation was made by Breiman in his seminal article (Breiman, 2001) introducing random forests. Note that the averaging procedure above is ensemble model combination and *not* Bayesian model averaging.

In the online learning setting, the training examples are presented one after another in a sequence of trials. Mondrian forests excel in this setting: at iteration $N+1$, each Mondrian tree $\mathcal{T} \sim \mathrm{MT}(\lambda, \mathcal{D}_{1:N})$ is updated to incorporate the next labeled example $(\boldsymbol{x}_{N+1}, y_{N+1})$ by sampling an extended tree $\mathcal{T}'$ from a distribution $\mathrm{MTx}(\lambda, \mathcal{T}, \mathcal{D}_{N+1})$. Using properties of the Mondrian process, we can choose a probability distribution MTx such that $\mathcal{T}' = \mathcal{T}$ on $\mathcal{D}_{1:N}$ and $\mathcal{T}'$ is distributed according to $\mathrm{MT}(\lambda, \mathcal{D}_{1:N+1})$, i.e.,

$$\begin{aligned} \mathcal{T} &\sim \mathrm{MT}(\lambda, \mathcal{D}_{1:N}) \\ \mathcal{T}' \mid \mathcal{T}, \mathcal{D}_{1:N+1} &\sim \mathrm{MTx}(\lambda, \mathcal{T}, \mathcal{D}_{N+1}) \end{aligned} \qquad \textit{implies} \qquad \mathcal{T}' \sim \mathrm{MT}(\lambda, \mathcal{D}_{1:N+1})\,. \tag{5.1}$$

Therefore, the distribution of Mondrian trees trained on a dataset in an incremental fashion is the same as that of Mondrian trees trained on the same dataset in a batch fashion, irrespective of the order in which the data points are observed. To the best of our knowledge, none of the existing online random forests have this property. Moreover, we can sample from $\mathrm{MTx}(\lambda, \mathcal{T}, \mathcal{D}_{N+1})$ efficiently: the complexity scales with the depth of the tree, which is typically logarithmic[1] in N.

While treating the online setting as a sequence of larger and larger batch problems is normally computationally prohibitive, this approach can be achieved efficiently with Mondrian forests. In the following sections, we define the Mondrian tree distribution $\mathrm{MT}(\lambda, \mathcal{D}_{1:N})$, the label distribution $p_{\mathcal{T}}(y|\boldsymbol{x}, \mathcal{D}_{1:N})$, and the update distribution $\mathrm{MTx}(\lambda, \mathcal{T}, \mathcal{D}_{N+1})$.

5.3 Mondrian trees

We refer to Section 2.2 and Figure 2.1 for a review of decision trees and our notation.

[1] See Section 5.7.2 for empirical tree depth results.

5.3.1 Mondrian process distribution over decision trees

Mondrian processes, introduced by Roy and Teh (2009), are families $\{\mathcal{M}_t : t \in [0, \infty)\}$ of random, hierarchical binary partitions of $\mathcal{X}$ such that $\mathcal{M}_t$ is a refinement of $\mathcal{M}_s$ whenever $t > s$.[2] Mondrian processes are natural candidates for the partition structure of random decision trees, but Mondrian processes on $\mathcal{X}$ are, in general, infinite structures that we cannot represent all at once. Because we only care about the partition on a finite set of observed data, we introduce **Mondrian trees**, which are restrictions of Mondrian processes to a finite set of points. A Mondrian tree T can be represented by a tuple $(\mathsf{T}, \boldsymbol{\delta}, \boldsymbol{\xi}, \boldsymbol{\tau})$, where $(\mathsf{T}, \boldsymbol{\delta}, \boldsymbol{\xi})$ is a decision tree and $\boldsymbol{\tau} = \{\tau_j\}_{j \in \mathsf{T}}$ associates a time of split $\tau_j \geq 0$ with each node j. Split times increase with depth, i.e., $\tau_j > \tau_{\mathsf{parent}(j)}$. We abuse notation and define $\tau_{\mathsf{parent}(\epsilon)} = 0$.

Given a non-negative *lifetime* parameter λ and training data $\mathcal{D}_{1:n}$, the generative process for sampling Mondrian trees from $\mathrm{MT}(\lambda, \mathcal{D}_{1:n})$ is described in the following two algorithms:

Algorithm 5.1 $\mathsf{SampleMondrianTree}(\lambda, \mathcal{D}_{1:n})$

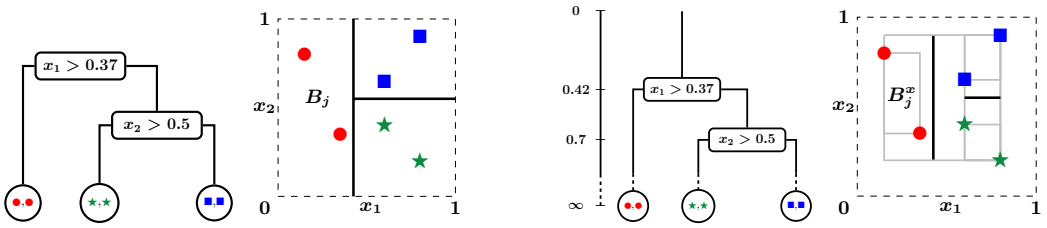
Initialize: $\mathsf{T} = \emptyset$, $\mathsf{leaves}(\mathsf{T}) = \emptyset$, $\boldsymbol{\delta} = \emptyset$, $\boldsymbol{\xi} = \emptyset$, $\boldsymbol{\tau} = \emptyset$, $N_\epsilon = \{1, 2, \ldots, n\}$
$\mathsf{SampleMondrianBlock}(\epsilon, \mathcal{D}_{N_\epsilon}, \lambda)$ ▷ *Algorithm 5.2*

Algorithm 5.2 $\mathsf{SampleMondrianBlock}(j, \mathcal{D}_{N_j}, \lambda)$

Add j to T
For all d, set $\ell^x_{jd} = \mathsf{min}(\boldsymbol{X}_{N_j,d}), u^x_{jd} = \mathsf{max}(\boldsymbol{X}_{N_j,d})$ ▷ *dimension-wise* min *and* max
Sample E from exponential distribution with rate $\sum_d (u^x_{jd} - \ell^x_{jd})$
if $\tau_{\mathsf{parent}(j)} + E < \lambda$ **then** ▷ *j is an internal node*
    Set $\tau_j = \tau_{\mathsf{parent}(j)} + E$
    Sample split dimension δ_j, choosing d with probability proportional to $u^x_{jd} - \ell^x_{jd}$
    Sample split location ξ_j uniformly from interval $[\ell^x_{j\delta_j}, u^x_{j\delta_j}]$
    Set $N_{\mathsf{left}(j)} = \{n \in N_j : \boldsymbol{X}_{n,\delta_j} \leq \xi_j\}$ and $N_{\mathsf{right}(j)} = \{n \in N_j : \boldsymbol{X}_{n,\delta_j} > \xi_j\}$
    $\mathsf{SampleMondrianBlock}(\mathsf{left}(j), \mathcal{D}_{N_{\mathsf{left}(j)}}, \lambda)$
    $\mathsf{SampleMondrianBlock}(\mathsf{right}(j), \mathcal{D}_{N_{\mathsf{right}(j)}}, \lambda)$
else ▷ *j is a leaf node*
    Set $\tau_j = \lambda$ and add j to $\mathsf{leaves}(\mathsf{T})$

The procedure starts with the root node ϵ and recurses down the tree. In Algorithm 5.2, we first compute the $\boldsymbol{\ell}^x_\epsilon$ and $\mathbf{u}^x_\epsilon$ i.e. the lower and upper bounds of B^x_ϵ, the smallest rectangle enclosing $\boldsymbol{X}_{N_\epsilon}$. We sample E from an exponential distribution whose rate is the so-called linear dimension of B^x_ϵ, given by $\sum_d (u^x_{\epsilon d} - \ell^x_{\epsilon d})$. Since $\tau_{\mathsf{parent}(\epsilon)} = 0$, $E + \tau_{\mathsf{parent}(\epsilon)} = E$. If $E \geq \lambda$, the time of split is not within the lifetime λ; hence, we assign ϵ to be a leaf node and the procedure halts. (Since $\mathbb{E}[E] = 1/\big(\sum_d (u^x_{jd} - \ell^x_{jd})\big)$,

[2] Roy and Teh (Roy and Teh, 2009) studied the distribution of $\{\mathcal{M}_t : t \leq \lambda\}$ and referred to λ as the *budget*. See (Roy, Chp. 5) for more details. We will refer to t as time, not be confused with discrete time in the online learning setting.



(a) Decision Tree (b) Mondrian Tree

Figure 5.1: Example of a decision tree in $[0,1]^2$ where x_1 and x_2 denote horizontal and vertical axis respectively: Figure 5.1(a) shows tree structure and partition of a decision tree, while Figure 5.1(b) shows a Mondrian tree. Note that the Mondrian tree is embedded on a vertical time axis, with each node associated with a time of split and the splits are committed only within the range of the training data in each block (denoted by gray rectangles). Let j denote the left child of the root: $B_j = (0, 0.37] \times (0, 1]$ denotes the block associated with red circles and $B_j^x \subseteq B_j$ is the smallest rectangle enclosing the two data points.

bigger rectangles are less likely to be leaf nodes.) Else, ϵ is an internal node and we sample a split $(\delta_\epsilon, \xi_\epsilon)$ from the *uniform split distribution* on B_ϵ^x. More precisely, we first sample the dimension δ_ϵ, taking the value d with probability proportional to $u_{\epsilon d}^x - \ell_{\epsilon d}^x$, and then sample the split location ξ_ϵ uniformly from the interval $[\ell_{\epsilon\delta_\epsilon}^x, u_{\epsilon\delta_\epsilon}^x]$. The procedure then recurses along the left and right children.

Mondrian trees differ from standard decision trees (e.g. CART, C4.5) in the following ways: (i) the splits are sampled independent of the labels Y_{N_j}; (ii) every node j is associated with a split time denoted by τ_j; (iii) the lifetime parameter λ controls the total number of splits (similar to the maximum depth parameter for standard decision trees); (iv) the split represented by an internal node j holds only within B_j^x and not the whole of B_j. No commitment is made in $B_j \setminus B_j^x$. Figure 5.1 illustrates the difference between decision trees and Mondrian trees.

Consider the family of distributions $\mathrm{MT}(\lambda, F)$, where F ranges over all possible finite sets of data points. Due to the fact that these distributions are derived from that of a Mondrian process on $\mathcal{X}$ restricted to a set F of points, the family $\mathrm{MT}(\lambda, \cdot)$ will be *projective*. Intuitively, projectivity implies that the tree distributions possess a type of self-consistency. In words, if we sample a Mondrian tree T from $\mathrm{MT}(\lambda, F)$ and then restrict the tree T to a subset $F' \subseteq F$ of points, then the restricted tree T' has distribution $\mathrm{MT}(\lambda, F')$. Most importantly, projectivity gives us a consistent way to extend a Mondrian tree on a data set $\mathcal{D}_{1:N}$ to a larger data set $\mathcal{D}_{1:N+1}$. We exploit this property to incrementally grow a Mondrian tree: we instantiate the Mondrian tree on the observed training data points; upon observing a new data point $\mathcal{D}_{N+1}$, we *extend* the Mondrian tree by sampling from the conditional distribution of a Mondrian tree on $\mathcal{D}_{1:N+1}$ given its restriction to $\mathcal{D}_{1:N}$, denoted by $\mathrm{MTx}(\lambda, \mathcal{T}, \mathcal{D}_{N+1})$ in (5.1). Thus, a Mondrian process on $\mathcal{X}$ is represented only where we have observed training data.

5.4 Label distribution: model, hierarchical prior, and predictive posterior

So far, our discussion has been focused on the tree structure. In this section, we focus on the predictive label distribution, $p_{\mathcal{T}}(y|\boldsymbol{x}, \mathcal{D}_{1:N})$, for a tree $\mathcal{T} = (\mathsf{T}, \boldsymbol{\delta}, \boldsymbol{\xi}, \boldsymbol{\tau})$, dataset $\mathcal{D}_{1:N}$, and test point $\boldsymbol{x}$. Let $\mathsf{leaf}(\boldsymbol{x})$ denote the unique leaf node $j \in \mathsf{leaves}(\mathsf{T})$ such that $\boldsymbol{x} \in B_j$. Intuitively, we want the predictive label distribution at $\boldsymbol{x}$ to be a smoothed version of the empirical distribution of labels for points in $B_{\mathsf{leaf}(\boldsymbol{x})}$ and in $B_{j'}$ for nearby nodes j'. We achieve this smoothing via a hierarchical Bayesian approach: every node is associated with a label distribution, and a prior is chosen under which the label distribution of a node is similar to that of its parent's. The predictive $p_{\mathcal{T}}(y|\boldsymbol{x}, \mathcal{D}_{1:N})$ is then obtained via marginalization.

As is common in the decision tree literature, we assume the labels within each block are independent of $\boldsymbol{X}$ given the tree structure. For every $j \in \mathsf{T}$, let G_j denote the distribution of labels at node j, and let $\mathcal{G} = \{G_j : j \in \mathsf{T}\}$ be the set of label distributions at all the nodes in the tree. Given $\mathcal{T}$ and $\mathcal{G}$, the predictive label distribution at $\boldsymbol{x}$ is $p(y|\boldsymbol{x}, \mathcal{T}, \mathcal{G}) = G_{\mathsf{leaf}(\boldsymbol{x})}$, i.e., the label distribution at the node $\mathsf{leaf}(\boldsymbol{x})$. In this chapter, we focus on the case of categorical labels taking values in the set $\{1, \ldots, K\}$, and so we abuse notation and write $G_{j,k}$ for the probability that a point in B_j is labeled k.

We model the collection G_j, for $j \in \mathsf{T}$, as a hierarchy of normalized stable processes (NSP) (Wood et al., 2009). A NSP prior is a distribution over distributions and is a special case of the Pitman-Yor process (PYP) prior where the concentration parameter is taken to zero (Pitman, 2006).[3] The discount parameter $d \in (0, 1)$ controls the variation around the base distribution; if $G_j \sim \mathrm{NSP}(d, H)$, then $\mathbb{E}[G_{jk}] = H_k$ and $\mathrm{Var}[G_{jk}] = (1 - d)H_k(1 - H_k)$. We use a hierarchical NSP (HNSP) prior over G_j as follows:

$$G_\epsilon | H \sim \mathrm{NSP}(d_\epsilon, H), \qquad \text{and} \qquad G_j | G_{\mathsf{parent}(j)} \sim \mathrm{NSP}(d_j, G_{\mathsf{parent}(j)}). \tag{5.2}$$

This hierarchical prior was first proposed by Wood et al. (2009). Here we take the base distribution H to be the uniform distribution over the K labels, and set $d_j = \exp\big(-\gamma(\tau_j - \tau_{\mathsf{parent}(j)})\big)$.

Given training data $\mathcal{D}_{1:N}$, the predictive distribution $p_{\mathcal{T}}(y|\boldsymbol{x}, \mathcal{D}_{1:N})$ is obtained by integrating over $\mathcal{G}$, i.e.,

$$p_{\mathcal{T}}(y|\boldsymbol{x}, \mathcal{D}_{1:N}) = \mathbb{E}_{\mathcal{G} \sim p_{\mathcal{T}}(\mathcal{G}|\mathcal{D}_{1:N})}[G_{\mathsf{leaf}(\boldsymbol{x}),y}] = \overline{G}_{\mathsf{leaf}(\boldsymbol{x}),y}, \tag{5.3}$$

[3] Taking the discount parameter to zero leads to a Dirichlet process . Hierarchies of NSPs admit more tractable approximations than hierarchies of Dirichlet processes (Wood et al., 2009), hence our choice here.

where the posterior over the label distributions is given by

$$p_{\mathcal{T}}(\mathcal{G}|\mathcal{D}_{1:N}) \propto p_{\mathcal{T}}(\mathcal{G}) \prod_{n=1}^{N} G_{\mathsf{leaf}(\boldsymbol{x}_n),y_n}. \tag{5.4}$$

Posterior inference in the HNSP, i.e., computation of the posterior means $\overline{G}_{\mathsf{leaf}(\boldsymbol{x})}$, is a special case of posterior inference in the hierarchical PYP (HPYP). In particular, Teh (2006) considers the HPYP with multinomial likelihood (in the context of language modeling). The model considered here is a special case of (Teh, 2006). Exact inference is intractable and hence we resort to approximations. In particular, we use a fast approximation known as the interpolated Kneser-Ney (IKN) smoothing (Teh, 2006), a popular technique for smoothing probabilities in language modeling (Goodman, 2001). The IKN approximation in (Teh, 2006) can be extended in a straightforward fashion to the online setting, and the computational complexity of adding a new training instance is linear in the depth of the tree. We present a detailed description of the posterior updates below for the interested reader.

5.4.1 Detailed description of posterior inference using the HNSP

Recall that we use a hierarchical Bayesian approach to specify a smooth label distribution $p_{\mathcal{T}}(y|\boldsymbol{x}, \mathcal{D}_{1:N})$ for each tree $\mathcal{T}$. The label prediction at a test point $\boldsymbol{x}$ will depend on where $\boldsymbol{x}$ falls relative to the existing data in the tree $\mathcal{T}$. In this section, we assume that $\boldsymbol{x}$ lies within one of the leaf nodes in $\mathcal{T}$, i.e., $\boldsymbol{x} \in B^x_{\mathsf{leaf}(\boldsymbol{x})}$, where $\mathsf{leaf}(\boldsymbol{x}) \in \mathsf{leaves}(\mathsf{T})$. If $\boldsymbol{x}$ does not lie within any of the leaf nodes in $\mathcal{T}$, i.e., $\boldsymbol{x} \notin \cup_{j \in \mathsf{leaves}(\mathsf{T})} B^x_j$, one could extend the tree by sampling $\mathcal{T}'$ from $\mathrm{MTx}(\lambda, \mathcal{T}, \boldsymbol{x})$, such that $\boldsymbol{x}$ lies within a leaf node in $\mathcal{T}'$ and apply the procedure described below using the extended tree $\mathcal{T}'$. Section 5.5.3 describes this case in more detail.

Given training data $\mathcal{D}_{1:N}$, a Mondrian tree $\mathcal{T}$ and the hierarchical prior over $\mathcal{G}$, the predictive label distribution $p_{\mathcal{T}}(y|\boldsymbol{x}, \mathcal{D}_{1:N})$ is obtained by integrating over $\mathcal{G}$, i.e.

$$p_{\mathcal{T}}(y|\boldsymbol{x}, \mathcal{D}_{1:N}) = \mathbb{E}_{\mathcal{G} \sim p_{\mathcal{T}}(\mathcal{G}|\mathcal{D}_{1:N})}[G_{\mathsf{leaf}(\boldsymbol{x}),y}] = \overline{G}_{\mathsf{leaf}(\boldsymbol{x}),y}.$$

Hence, the prediction is given by $\overline{G}_{\mathsf{leaf}(\boldsymbol{x})}$, the posterior mean at $\mathsf{leaf}(\boldsymbol{x})$. The posterior mean $\overline{G}_{\mathsf{leaf}(\boldsymbol{x})}$ can be computed using existing techniques, which we review in the rest of this section.

Posterior inference in the HNSP is a special case of posterior inference in hierarchical PYP (HPYP). Teh (2006) considers the HPYP with multinomial likelihood (in the context of language modeling)—the model considered here (HNSP with multinomial likelihood) is a special case of (Teh, 2006). Hence, we just sketch the high level picture and refer the reader to (Teh, 2006) for further details. We first describe posterior inference given N data points $\mathcal{D}_{1:N}$ (batch setting), and later explain how to adapt

inference to the online setting. Finally, we describe the computation of the predictive posterior distribution.

Batch setting

Posterior inference is done using the Chinese restaurant process representation, wherein every node of the decision tree is a restaurant; the training data points are the customers seated in the tables associated with the leaf node restaurants; these tables are in turn customers at the tables in their corresponding parent level restaurant; the dish served at each table is the class label. Exact inference is intractable and hence we resort to approximations. In particular, we use the approximation known as the interpolated Kneser-Ney (IKN) smoothing, a popular smoothing technique for language modeling (Goodman, 2001). The IKN smoothing can be interpreted as an approximate inference scheme for the HPYP, where the number of tables serving a particular dish in a restaurant is at most one (Teh, 2006). More precisely, if $c_{j,k}$ denotes the number of customers at restaurant j eating dish k and $\mathsf{tab}_{j,k}$ denotes the number of tables at restaurant j serving dish k, the IKN approximation sets $\mathsf{tab}_{j,k} = \mathsf{min}(c_{j,k}, 1)$. The counts $c_{j,k}$ and $\mathsf{tab}_{j,k}$ can be computed in a single bottom-up pass as follows: for every leaf node $j \in \mathsf{leaves}(\mathsf{T})$, $c_{j,k}$ is simply the number of training data points with label k at node j; for every internal node $j \in \mathsf{T} \setminus \mathsf{leaves}(\mathsf{T})$, we set $c_{j,k} = \mathsf{tab}_{\mathsf{left}(j),k} + \mathsf{tab}_{\mathsf{right}(j),k}$. For a leaf node j, this procedure is summarized in Algorithm 5.3. (Note that this pseudocode just serves as a reference; in practice, these counts are updated in an online fashion, as described in Algorithm 5.7.)

Algorithm 5.3 InitializePosteriorCounts(j)

For all k, set $c_{jk} = \#\{n \in N_j : y_n = k\}$
Initialize $j' = j$
while True **do**
    if $j' \notin \mathsf{leaves}(\mathsf{T})$ **then**
        For all k, set $c_{j'k} = \mathsf{tab}_{\mathsf{left}(j'),k} + \mathsf{tab}_{\mathsf{right}(j'),k}$
    For all k, set $\mathsf{tab}_{j'k} = \mathsf{min}(c_{j'k}, 1)$ ▷ *IKN approximation*
    if $j' = \epsilon$ **then**
        return
    else
        $j' \leftarrow \mathsf{parent}(j')$

Predictive posterior computation

Given the counts $c_{j,k}$ and table assignments $\mathsf{tab}_{j,k}$, the predictive probability (i.e., posterior mean) at node j can be computed recursively as follows:

$$\overline{G}_{jk} = \begin{cases} \dfrac{c_{j,k} - d_j \mathsf{tab}_{j,k}}{c_{j,\cdot}} + \dfrac{d_j \mathsf{tab}_{j,\cdot}}{c_{j,\cdot}} \, \overline{G}_{\mathsf{parent}(j),k} & c_{j,\cdot} > 0, \\ \overline{G}_{\mathsf{parent}(j),k} & c_{j,\cdot} = 0, \end{cases} \tag{5.5}$$

where $c_{j,\cdot} = \sum_k c_{j,k}$, $\mathsf{tab}_{j,\cdot} = \sum_k \mathsf{tab}_{j,k}$, and $d_j := \exp\big(-\gamma(\tau_j - \tau_{\mathsf{parent}(j)})\big)$ is the *discount* for node j, defined in Section 5.4. Informally, the discount interpolates between the counts c and the prior. If the discount $d_j \approx 1$, then $\overline{G}_j$ is more like its parent $\overline{G}_{\mathsf{parent}(j)}$. If $d_j \approx 0$, then $\overline{G}_j$ weights the counts more. These predictive probabilities can be computed in a single top-down pass as shown in Algorithm 5.4.

Algorithm 5.4 ComputePosteriorPredictiveDistribution$\big(\mathcal{T}, \mathcal{G}\big)$

▷ *Description of top-down pass to compute posterior predictive distribution given by (5.5)*
▷ $\overline{G}_{jk}$ *denotes the posterior probability of* $y = k$ *at node* j
Initialize the ordered set $J = \{\epsilon\}$
while J not empty **do**
    Pop the first element of J
    if $j = \epsilon$ **then**
        $\overline{G}_{\mathsf{parent}(\epsilon)} = H$
    Set $d = \exp\big(-\gamma(\tau_j - \tau_{\mathsf{parent}(j)})\big)$
    For all k, set $\overline{G}_{jk} = c_{j,\cdot}^{-1}\Big(c_{j,k} - d\ \mathsf{tab}_{j,k} + d\ \mathsf{tab}_{j,\cdot}\ \overline{G}_{\mathsf{parent}(j),k}\Big)$
    if $j \notin \mathsf{leaves}(\mathsf{T})$ **then**
        Append $\mathsf{left}(j)$ and $\mathsf{right}(j)$ to the end of the ordered set J

5.5 Online training and prediction

In this section, we describe the family of distributions $\mathrm{MTx}(\lambda, \mathcal{T}, \mathcal{D}_{N+1})$, which are used to incrementally add a data point, $\mathcal{D}_{N+1}$, to a tree $\mathcal{T}$. These updates are based on the conditional Mondrian algorithm (Roy and Teh, 2009), specialized to a finite set of points. In general, one or more of the following three operations may be executed while introducing a new data point: (i) introduction of a new split 'above' an existing split, (ii) extension of an existing split to the updated extent of the block and (iii) splitting an existing leaf node into two children. To the best of our knowledge, existing online decision trees use just the third operation, and the first two operations are unique to Mondrian trees. The complete pseudo-code for incrementally updating a Mondrian tree $\mathcal{T}$ with a new data point $\mathcal{D}$ according to $\mathrm{MTx}(\lambda, \mathcal{T}, \mathcal{D})$ is described in the following two algorithms. Figure 5.2 walks through the algorithms on a toy dataset.

Algorithm 5.5 ExtendMondrianTree$(\mathcal{T}, \lambda, \mathcal{D})$

Input: Tree $\mathcal{T} = (\mathsf{T}, \boldsymbol{\delta}, \boldsymbol{\xi}, \boldsymbol{\tau})$, new training instance $\mathcal{D} = (\boldsymbol{x}, y)$
ExtendMondrianBlock$(\mathcal{T}, \lambda, \epsilon, \mathcal{D})$ ▷ *Algorithm 5.6*

Algorithm 5.6 ExtendMondrianBlock$(\mathcal{T}, \lambda, j, \mathcal{D})$

Set $\mathbf{e}^{\ell} = \max(\boldsymbol{\ell}_j^x - \boldsymbol{x}, 0)$ and $\mathbf{e}^{u} = \max(\boldsymbol{x} - \mathbf{u}_j^x, 0)$ ▷ $\mathbf{e}^{\ell} = \mathbf{e}^{u} = \mathbf{0}_D$ *if* $\boldsymbol{x} \in B_j^x$
Sample E from exponential distribution with rate $\sum_d (e_d^{\ell} + e_d^{u})$
if $\tau_{\mathsf{parent}(j)} + E < \tau_j$ **then** ▷ *introduce new parent for node j*
    Sample split dimension δ, choosing d with probability proportional to $e_d^{\ell} + e_d^{u}$
    Sample split location ξ uniformly from interval $[u_{j,\delta}^x, x_\delta]$ **if** $x_\delta > u_{j,\delta}^x$ **else** $[x_\delta, \ell_{j,\delta}^x]$.
    Insert a new node $\tilde{j}$ just above node j in the tree, and a new leaf j'', sibling to j, where $\delta_{\tilde{j}} = \delta$, $\xi_{\tilde{j}} = \xi$, $\tau_{\tilde{j}} = \tau_{\mathsf{parent}(j)} + E$, $\boldsymbol{\ell}_{\tilde{j}}^x = \min(\boldsymbol{\ell}_j^x, \boldsymbol{x})$, $\mathbf{u}_{\tilde{j}}^x = \max(\mathbf{u}_j^x, \boldsymbol{x})$
    $j'' = \mathsf{left}(\tilde{j})$ **iff** $x_{\delta_{\tilde{j}}} \leq \xi_{\tilde{j}}$
    SampleMondrianBlock$(j'', \mathcal{D}, \lambda)$
else
    Update $\boldsymbol{\ell}_j^x \leftarrow \min(\boldsymbol{\ell}_j^x, \boldsymbol{x}), \mathbf{u}_j^x \leftarrow \max(\mathbf{u}_j^x, \boldsymbol{x})$ ▷ *update extent of node j*
    if $j \notin \mathsf{leaves}(\mathsf{T})$ **then** ▷ *return if j is a leaf node, else recurse down the tree*
        if $x_{\delta_j} \leq \xi_j$ **then** $\mathsf{child}(j) = \mathsf{left}(j)$ **else** $\mathsf{child}(j) = \mathsf{right}(j)$
        ExtendMondrianBlock$(\mathcal{T}, \lambda, \mathsf{child}(j), \mathcal{D})$ ▷ *recurse on child containing* $\mathcal{D}$

5.5.1 Controlling Mondrian tree complexity

The most common strategies for controlling tree complexity in random forests are controlling the maximum depth or controlling the number of data points required to split a node. The Mondrian tree complexity parameter λ is analogous to depth, however it does not allow us to control the total number of nodes in the tree. Hence it is not straightforward to specify λ, especially in the online setting. Hence, in this chapter as well as Chapter 6, we focus on controlling the minimum number of data points before a node is split. For classification problems, we set $\mathsf{min\_samples\_split} = 2$ following (Geurts et al., 2006). In practice, random forest implementations also stop splitting a node when all the labels are identical and assign it to be a leaf node. In the online learning setting, the label distributions can change with time. To make our MF implementation comparable[4] with the corresponding batch RF version, we '*pause*' a Mondrian block when all the labels are identical; if a new training instance lies within B_j of a paused leaf node j and has the same label as the rest of the data points in B_j, we continue pausing the Mondrian block. We '*un-pause*' the Mondrian block when there is more than one unique label in that block. Algorithms 5.9 and 5.10 in section 5.5.4 discuss versions of SampleMondrianBlock and ExtendMondrianBlock for paused Mondrians.

[4]Specifically, pausing provides computational speedup for MFs. Controlling tree complexity is important to prevent over-fitting in Breiman-RF and ERT; MFs use hierarchical smoothing and hence are less prone to over-fitting compared to Breiman-RF and ERT. However, pausing ensures that that MF and ERT-1 contain comparable number of leaves.

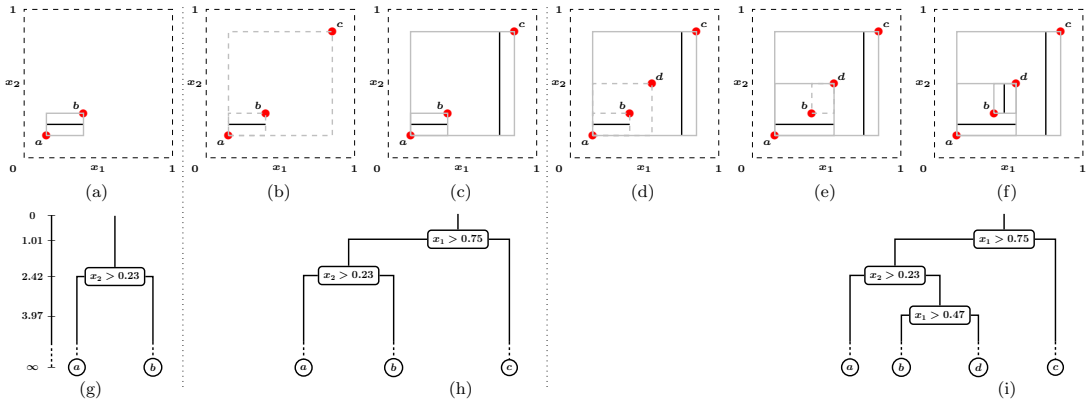


Figure 5.2: Online learning with Mondrian trees on a toy dataset: We assume that $\lambda = \infty, D = 2$ and add one data point at each iteration. For simplicity, we ignore class labels and denote location of training data with red circles. Figures 2(a), 2(c) and 2(f) show the partitions after the first, second and third iterations, respectively, with the intermediate figures denoting intermediate steps. Figures 2(g), 2(h) and 2(i) show the trees after the first, second and third iterations, along with a shared vertical time axis.

At iteration 1, we have two training data points, labeled as a, b. Figures 2(a) and 2(g) show the partition and tree structure of the Mondrian tree. Note that even though there is a split $x_2 > 0.23$ at time $t = 2.42$, we commit this split only within B_j^x (shown by the gray rectangle).

At iteration 2, a new data point c is added. Algorithm 5.5 starts with the root node and recurses down the tree. Algorithm 5.6 checks if the new data point lies within B_ϵ^x by computing the additional extent $\mathbf{e}^\ell$ and $\mathbf{e}^u$. In this case, c does not lie within B_ϵ^x. Let R_{ab} and R_{abc} respectively denote the small gray rectangle (enclosing a, b) and big gray rectangle (enclosing a, b, c) in Figure 2(b). While extending the Mondrian from R_{ab} to R_{abc}, we could either introduce a new split in R_{abc} outside R_{ab} or extend the split in R_{ab} to the new range. To choose between these two options, we sample the time of this new split: we first sample E from an exponential distribution whose rate is the sum of the additional extent, i.e., $\sum_d (e_d^\ell + e_d^u)$, and set the time of the new split to $E + \tau_{\mathsf{parent}(\epsilon)}$. If $E + \tau_{\mathsf{parent}(\epsilon)} \leq \tau_\epsilon$, this new split in R_{abc} can precede the old split in R_{ab} and a split is sampled in R_{abc} outside R_{ab}. In Figures 2(c) and 2(h), $E + \tau_{\mathsf{parent}(\epsilon)} = 1.01 + 0 \leq 2.42$, hence a new split $x_1 > 0.75$ is introduced. The farther a new data point $\boldsymbol{x}$ is from B_j^x, the higher the rate $\sum_d (e_d^\ell + e_d^u)$, and subsequently the higher the probability of a new split being introduced, since $\mathbb{E}[E] = 1/\big(\sum_d (e_d^\ell + e_d^u)\big)$. A new split in R_{abc} is sampled such that it is consistent with the existing partition structure in R_{ab} (i.e., the new split cannot slice through R_{ab}).

In the final iteration, we add data point d. In Figure 2(d), the data point d lies within the extent of the root node, hence we traverse to the left side of the root and update B_j^x of the internal node containing $\{a, b\}$ to include d. We could either introduce a new split or extend the split $x_2 > 0.23$. In Figure 2(e), we extend the split $x_2 > 0.23$ to the new extent, and traverse to the leaf node in Figure 2(h) containing b. In Figures 2(f) and 2(i), we sample $E = 1.55$ and since $\tau_{\mathsf{parent}(j)} + E = 2.42 + 1.55 = 3.97 \leq \lambda = \infty$, we introduce a new split $x_1 > 0.47$.

5.5.2 Posterior inference: online setting

It is straightforward to extend inference to the online setting. Adding a new data point $\mathcal{D} = (\boldsymbol{x}, y)$ affects only the counts along the path from the root to the leaf node of that data point. We update the counts in a bottom-up fashion, starting at the leaf node containing the data point, $\mathsf{leaf}(\boldsymbol{x})$. Due to the nature of the IKN approximation, we can stop at the internal node j where $c_{j,y} = 1$ and need not traverse up till the root. This procedure is summarized in Algorithm 5.7.

Algorithm 5.7 UpdatePosteriorCounts(j, y)

$c_{jy} \leftarrow c_{jy} + 1$
Initialize $j' = j$
while True **do**
  if $\mathsf{tab}_{j'y} = 1$ **then** ▷ *none of the counts above need to be updated*
    return
  else
    if $j' \notin \mathsf{leaves}(\mathsf{T})$ **then**
      $c_{j'y} = \mathsf{tab}_{\mathsf{left}(j'),y} + \mathsf{tab}_{\mathsf{right}(j'),y}$
    $\mathsf{tab}_{j'y} = \mathsf{min}(c_{j'y}, 1)$ ▷ *IKN approximation*
    if $j' = \epsilon$ **then**
      return
    else
      $j' \leftarrow \mathsf{parent}(j')$

5.5.3 Prediction using Mondrian tree

Let $\boldsymbol{x}$ denote a test data point. If $\boldsymbol{x}$ is already 'contained' in the tree $\mathcal{T}$, i.e., if $\boldsymbol{x} \in B_j^x$ for some leaf $j \in \mathsf{leaves}(\mathsf{T})$, then the prediction is taken to be $\overline{G}_{\mathsf{leaf}(\boldsymbol{x})}$. Otherwise, we somehow need to incorporate $\boldsymbol{x}$. One choice is to extend $\mathcal{T}$ by sampling $\mathcal{T}'$ from $\mathrm{MTx}(\lambda, \mathcal{T}, \boldsymbol{x})$ as described in Algorithm 5.5, and set the prediction to $\overline{G}_j$, where $j \in \mathsf{leaves}(\mathsf{T}')$ is the leaf node containing $\boldsymbol{x}$. A particular extension $\mathcal{T}'$ might lead to an overly confident prediction; hence, we average over *every* possible extension $\mathcal{T}'$. This integration can be carried out analytically and the computational complexity is linear in the depth of the tree. We provide a detailed description below for the interested reader.

Detailed description of prediction using Mondrian tree

Let $\boldsymbol{x}$ denote a test data point. We are interested in the predictive probability of y at $\boldsymbol{x}$, denoted by $p_{\mathcal{T}}(y|\boldsymbol{x}, \mathcal{D}_{1:N})$. As in typical decision trees, the process involves a top-down tree traversal, starting from the root. If $\boldsymbol{x}$ is already 'contained' in the tree $\mathcal{T}$, i.e., if $\boldsymbol{x} \in B_j^x$ for some leaf $j \in \mathsf{leaves}(\mathsf{T})$, then the prediction is taken to be $\overline{G}_{\mathsf{leaf}(\boldsymbol{x})}$, which is computed as described in Section 5.4.1. Otherwise, we somehow need to incorporate $\boldsymbol{x}$. One choice is to extend $\mathcal{T}$ by sampling $\mathcal{T}'$ from $\mathrm{MTx}(\lambda, \mathcal{T}, \boldsymbol{x})$ as

described in Algorithm 5.5, and set the prediction to $\overline{G}_j$, where $j \in \mathsf{leaves}(\mathsf{T}')$ is the leaf node containing $\boldsymbol{x}$. A particular extension $\mathcal{T}'$ might lead to an overly confident prediction; hence, we average over *every* possible extension $\mathcal{T}'$. This expectation can be carried out analytically, using properties of the Mondrian process, as we show below.

Let $\mathsf{ancestors}(j)$ denote the set of all ancestors of node j. Let $\mathsf{path}(j) = \{j\} \cup \mathsf{ancestors}(j)$, that is, the set of all nodes along the ancestral path from j to the root. Recall that $\mathsf{leaf}(\boldsymbol{x})$ is the unique leaf node in T such that $\boldsymbol{x} \in B_{\mathsf{leaf}(\boldsymbol{x})}$. If the test point $\boldsymbol{x} \in B^x_{\mathsf{leaf}(\boldsymbol{x})}$ (i.e., $\boldsymbol{x}$ lies within the 'gray rectangle' at the leaf node), it can never branch off; else, it can branch off at one or more points along the path from the root to $\mathsf{leaf}(\boldsymbol{x})$. More precisely, if $\boldsymbol{x}$ lies outside B^x_j at node j, the probability that $\boldsymbol{x}$ will branch off into its own node at node j, denoted by[5] $p^s_j(\boldsymbol{x})$, is equal to the probability that a split exists in B_j outside B^x_j, which is

$$p^s_j(\boldsymbol{x}) = 1 - \exp\big(-\Delta_j \eta_j(\boldsymbol{x})\big), \quad \text{where } \eta_j(\boldsymbol{x}) = \sum_d \big(\mathsf{max}(x_d - u^x_{jd}, 0) + \mathsf{max}(\ell^x_{jd} - x_d, 0)\big),$$

and $\Delta_j = \tau_j - \tau_{\mathsf{parent}(j)}$. Note that $p^s_j(\boldsymbol{x}) = 0$ if $\boldsymbol{x}$ lies within B^x_j (i.e., if $\ell^x_{jd} \le x_d \le u^x_{jd}$ for all d). The probability of $\boldsymbol{x}$ not branching off before reaching node j is given by $\prod_{j' \in \mathsf{ancestors}(j)} (1 - p^s_{j'}(\boldsymbol{x}))$.

If $\boldsymbol{x} \in B^x_{\mathsf{leaf}(\boldsymbol{x})}$, the prediction is given by $\overline{G}_{\mathsf{leaf}(\boldsymbol{x})}$. If there is a split in B_j outside B^x_j, let $\tilde{j}$ denote the new parent of j and $\mathsf{child}(\tilde{j})$ denote the child node containing just the test data point,; in this case, the prediction is $\overline{G}_{\mathsf{child}(\tilde{j})}$. Averaging over the location where the test point branches off, we obtain

$$p_{\mathcal{T}}(y|\boldsymbol{x}, \mathcal{D}_{1:N}) = \sum_{j \in \mathsf{path}(\mathsf{leaf}(\boldsymbol{x}))} \Big(\prod_{j' \in \mathsf{ancestors}(j)} (1 - p^s_{j'}(\boldsymbol{x})) \Big) F_j(\boldsymbol{x}), \tag{5.6}$$

where

$$F_j(\boldsymbol{x}) = p^s_j(\boldsymbol{x}) \mathbb{E}_{\Delta_{\tilde{j}}} \Big[\overline{G}_{\mathsf{child}(\tilde{j})} \Big] + \mathbb{1}[j = \mathsf{leaf}(\boldsymbol{x})](1 - p^s_j(\boldsymbol{x})) \overline{G}_{\mathsf{leaf}(\boldsymbol{x})}. \tag{5.7}$$

The second term in $F_j(\boldsymbol{x})$ needs to be computed only for the leaf node $\mathsf{leaf}(\boldsymbol{x})$ and is simply the posterior mean of $G_{\mathsf{leaf}(\boldsymbol{x})}$ weighted by $1 - p^s_{\mathsf{leaf}(\boldsymbol{x})}(\boldsymbol{x})$. The posterior mean of $G_{\mathsf{leaf}(x)}$, given by $\overline{G}_{\mathsf{leaf}(\boldsymbol{x})}$, can be computed using (5.5). The first term in $F_j(\boldsymbol{x})$ is simply the posterior mean of $G_{\mathsf{child}(\tilde{j})}$, averaged over $\Delta_{\tilde{j}}$, weighted by $p^s_j(\boldsymbol{x})$. Since no labels are observed in $\mathsf{child}(\tilde{j})$, $c_{\mathsf{child}(\tilde{j}),\cdot} = 0$, hence from (5.5), we have $\overline{G}_{\mathsf{child}(\tilde{j})} = \overline{G}_{\tilde{j}}$. We compute $\overline{G}_{\tilde{j}}$ using (5.5). We average over $\Delta_{\tilde{j}}$ due to the fact that the discount in (5.5) for the node $\tilde{j}$ depends on $\tau_{\tilde{j}} - \tau_{\mathsf{parent}(\tilde{j})} = \Delta_{\tilde{j}}$. To average over all valid split times $\tau_{\tilde{j}}$, we compute expectation w.r.t. $\Delta_{\tilde{j}}$ which is distributed according to a truncated exponential with rate $\eta_j(\boldsymbol{x})$, truncated to the interval $[0, \Delta_j]$.

The procedure for computing $p_{\mathcal{T}}(y|\boldsymbol{x}, \mathcal{D}_{1:N})$ for any $\boldsymbol{x} \in \mathbb{R}^D$ is summarized in Algo-

[5] The superscript s in $p^s_j(\boldsymbol{x})$ is used to denote the fact that this split 'separates' the test data point $\boldsymbol{x}$ into its own leaf node.

rithm 5.8. The predictive probability assigned by a Mondrian forest is the average of the predictive probability of the M trees, i.e., $\frac{1}{M}\sum_m p_{\mathcal{T}_m}(y|\boldsymbol{x}, \mathcal{D}_{1:N})$.

Algorithm 5.8 $\mathsf{Predict}(\mathcal{T}, \boldsymbol{x})$ (prediction using Mondrian classification tree)

▷ *Description of prediction using a Mondrian tree, given by (5.6)*
Initialize $j = \epsilon$ and $p_{\mathsf{NotSeparatedYet}} = 1$
Initialize $\mathbf{s} = \mathbf{0}_K$ ▷ $\mathbf{s}$ *is* K*-dimensional vector where* $s_k = p_{\mathcal{T}}(y = k|\boldsymbol{x}, \mathcal{D}_{1:N})$
while True **do**
  Set $\Delta_j = \tau_j - \tau_{\mathsf{parent}(j)}$ and $\eta_j(\boldsymbol{x}) = \sum_d \big(\mathsf{max}(x_d - u^x_{jd}, 0) + \mathsf{max}(\ell^x_{jd} - x_d, 0)\big)$
  Set $p^s_j(\boldsymbol{x}) = 1 - \exp\big(-\Delta_j \eta_j(\boldsymbol{x})\big)$
  if $p^s_j(\boldsymbol{x}) > 0$ **then**
    ▷ *Let* $\boldsymbol{x}$ *branch off into its own node* $\mathsf{child}(j)$*, creating a new node* j *which is the parent of* j *and* $\mathsf{child}(j)$*.* $\overline{G}_{\mathsf{child}(j)} = \overline{G}_j$ *from (5.5) since* $c_{\mathsf{child}(j),\cdot} = 0$*.*
    Compute expected discount $\bar{d} = \mathbb{E}_\Delta[\exp(-\gamma\Delta)]$ where Δ is drawn from a truncated exponential with rate $\eta_j(\boldsymbol{x})$, truncated to the interval $[0, \Delta_j]$.
    For all k, set $c_{j,k} = \mathsf{tab}_{j,k} = \mathsf{min}(c_{j,k}, 1)$
    For all k, set $\overline{G}_{jk} = c^{-1}_{j,\cdot}\Big(c_{j,k} - \bar{d}\ \mathsf{tab}_{j,k} + \bar{d}\ \mathsf{tab}_{j,\cdot}\ \overline{G}_{\mathsf{parent}(j),k}\Big)$ ▷ *Algorithm 5.4*
    For all k, update $s_k \leftarrow s_k + p_{\mathsf{NotSeparatedYet}}\ p^s_j(\boldsymbol{x})\overline{G}_{jk}$
  if $j \in \mathsf{leaves}(\mathsf{T})$ **then**
    For all k, update $s_k \leftarrow s_k + p_{\mathsf{NotSeparatedYet}}(1 - p^s_j(\boldsymbol{x}))\overline{G}_{jk}$ ▷ *Algorithm 5.4*
    return predictive probability $\mathbf{s}$ where $s_k = p_{\mathcal{T}}(y = k|\boldsymbol{x}, \mathcal{D}_{1:N})$
  else
    $p_{\mathsf{NotSeparatedYet}} \leftarrow p_{\mathsf{NotSeparatedYet}}(1 - p^s_j(\boldsymbol{x}))$
    if $x_{\delta_j} \leq \xi_j$ **then** $j \leftarrow \mathsf{left}(j)$ **else** $j \leftarrow \mathsf{right}(j)$ ▷ *recurse to child where* $\boldsymbol{x}$ *lies*

5.5.4 Pseudocode for paused Mondrians

In this section, we discuss versions of SampleMondrianBlock and ExtendMondrianBlock for paused Mondrians in Algorithms 5.9 and 5.10 respectively. For completeness, we also provide the updates necessary for the IKN approximation within Algorithms 5.9 and 5.10.

5.6 Related work

The literature on random forests is vast and we do not attempt to cover it comprehensively; we provide a brief review here and refer to (Criminisi et al., 2012) and (Denil et al., 2013) for a recent review of random forests in batch and online settings respectively. Classic decision tree induction procedures choose the best split dimension and location from all candidate splits at each node by optimizing some suitable quality criterion (e.g. information gain) in a greedy manner. In a random forest, the individual trees are randomized to de-correlate their predictions. The most common strategies for injecting randomness are (i) bagging (Breiman, 1996) and (ii) randomly subsampling the set of candidate splits within each node.

Algorithm 5.9 $\mathsf{SampleMondrianBlock}\big(j, \mathcal{D}_{N_j}, \lambda\big)$ version that depends on labels

Add j to T
For all d, set $\ell^x_{jd} = \mathsf{min}(\boldsymbol{X}_{N_j,d}), u^x_{jd} = \mathsf{max}(\boldsymbol{X}_{N_j,d})$ ▷ *dim-wise* min *and* max
if $\mathsf{AllLabelsIdentical}(Y_{N_j})$ **then**
    Set $\tau_j = \lambda$ ▷ *pause Mondrian*
else
    Sample E from exponential distribution with rate $\sum_d (u^x_{jd} - \ell^x_{jd})$
    Set $\tau_j = \tau_{\mathsf{parent}(j)} + E$
if $\tau_j < \lambda$ **then**
    Sample split dimension δ_j with probability of choosing d proportional to $u^x_{jd} - \ell^x_{jd}$
    Sample split location ξ_j along dimension δ_j from an uniform distribution over $\mathcal{U}[\ell^x_{jd}, u^x_{jd}]$
    Set $N_{\mathsf{left}(j)} = \{n \in N_j : \boldsymbol{X}_{n,\delta_j} \leq \xi_j\}$ and $N_{\mathsf{right}(j)} = \{n \in N_j : \boldsymbol{X}_{n,\delta_j} > \xi_j\}$
    $\mathsf{SampleMondrianBlock}\big(\mathsf{left}(j), \mathcal{D}_{N_{\mathsf{left}(j)}}, \lambda\big)$
    $\mathsf{SampleMondrianBlock}\big(\mathsf{right}(j), \mathcal{D}_{N_{\mathsf{right}(j)}}, \lambda\big)$
else
    Set $\tau_j = \lambda$ and add j to $\mathsf{leaves}(\mathsf{T})$ ▷ *j is a leaf node*
    $\mathsf{InitializePosteriorCounts}(j)$ ▷ *Algorithm 5.3*

Two popular random forest variants in the batch setting are *Breiman-RF* (Breiman, 2001) and *Extremely randomized trees (ERT)* (Geurts et al., 2006). Breiman-RF uses bagging and furthermore, at each node, a random k-dimensional subset of the original D features is sampled. ERT chooses a k dimensional subset of the features and then chooses one split location each for the k features randomly (unlike Breiman-RF which considers all possible split locations along a dimension). ERT does not use bagging. When $k = 1$, the ERT trees are *totally randomized* and the splits are chosen independent of the labels; hence the ERT-1 method is very similar to MF in the batch setting in terms of tree induction. (Note that unlike ERT, MF uses HNSP to smooth predictive estimates and allows a test point to branch off into its own node.) Perfect random trees (PERT), proposed by Cutler and Zhao (2001) for classification problems, produce totally randomized trees similar to ERT-1, although there are some slight differences (Geurts et al., 2006).

Existing online random forests (ORF-Saffari (Saffari et al., 2009) and ORF-Denil (Denil et al., 2013)) start with an empty tree and grow the tree incrementally. Every leaf of every tree maintains a list of k candidate splits and associated quality scores. When a new data point is added, the scores of the candidate splits at the corresponding leaf node are updated. To reduce the risk of choosing a sub-optimal split based on noisy quality scores, additional hyper parameters such as the minimum number of data points at a leaf node before a decision is made and the minimum threshold for the quality criterion of the best split, are used to assess 'confidence' associated with a split. Once these criteria are satisfied at a leaf node, the best split is chosen (making this node an internal node) and its two children are the new leaf nodes (with their own candidate splits), and the process is repeated. These methods could be memory inefficient for

Algorithm 5.10 ExtendMondrianBlock$(\mathcal{T}, \lambda, j, \mathcal{D})$ version that depends on labels

if AllLabelsIdentical(Y_{N_j}) **then** ▷ *paused Mondrian leaf*
  Update extent $\boldsymbol{\ell}_j^x \leftarrow \min(\boldsymbol{\ell}_j^x, \boldsymbol{x}), \mathbf{u}_j^x \leftarrow \max(\mathbf{u}_j^x, \boldsymbol{x})$
  Append $\mathcal{D}$ to $\mathcal{D}_{N_j}$ ▷ *append* $\boldsymbol{x}$ *to* X_{N_j} *and* y *to* Y_{N_j}
  if $y =$ unique(Y_{N_j}) **then**
    UpdatePosteriorCounts(j, y) ▷ *Algorithm 5.7*
    return ▷ *continue pausing*
  else
    Remove j from leaves(T)
    SampleMondrianBlock$\big(j, \mathcal{D}_{N_j}, \lambda\big)$ ▷ *un-pause Mondrian; Algorithm 5.9*
else
  Set $\mathbf{e}^\ell = \max(\boldsymbol{\ell}_j^x - \boldsymbol{x}, 0)$ and $\mathbf{e}^u = \max(\boldsymbol{x} - \mathbf{u}_j^x, 0)$ ▷ $\mathbf{e}^\ell = \mathbf{e}^u = \mathbf{0}_D$ *if* $\boldsymbol{x} \in B_j^x$
  Sample E from exponential distribution with rate $\sum_d (e_d^\ell + e_d^u)$
  if $\tau_{\text{parent}(j)} + E < \tau_j$ **then** ▷ *introduce new parent for node* j
    Create new Mondrian block $\tilde{j}$ where $\boldsymbol{\ell}_{\tilde{j}}^x = \min(\boldsymbol{\ell}_j^x, \boldsymbol{x})$ and $\mathbf{u}_{\tilde{j}}^x = \max(\mathbf{u}_j^x, \boldsymbol{x})$
    Sample $\delta_{\tilde{j}}$ with $\mathbb{P}(\delta_{\tilde{j}} = d)$ proportional to $e_d^\ell + e_d^u$
    if $x_{\delta_{\tilde{j}}} > u_{j,\delta_{\tilde{j}}}^x$, **then** sample $\xi_{\tilde{j}}$ from $\mathcal{U}[u_{j,\delta_{\tilde{j}}}^x, x_{\delta_{\tilde{j}}}]$,
    else sample $\xi_{\tilde{j}}$ from $\mathcal{U}([x_{\delta_{\tilde{j}}}, \ell_{j,\delta_{\tilde{j}}}^x])$
    if $j = \epsilon$ **then** ▷ *set* $\tilde{j}$ *as the new root*
      $\epsilon \leftarrow \tilde{j}$
    else ▷ *set* $\tilde{j}$ *as child of* parent(j)
      if $j =$ left(parent(j)), **then** left(parent(j)) $\leftarrow \tilde{j}$, **else** right(parent(j)) $\leftarrow \tilde{j}$
    if $x_{\delta_{\tilde{j}}} > \xi_{\tilde{j}}$ **then**
      Set left$(\tilde{j}) = j$ and SampleMondrianBlock$\big($right$(\tilde{j}), \mathcal{D}, \lambda\big)$ ▷ *create new leaf*
    else
      Set right$(\tilde{j}) = j$ and SampleMondrianBlock$\big($left$(\tilde{j}), \mathcal{D}, \lambda\big)$ ▷ *create new leaf*
  else
    Update $\boldsymbol{\ell}_j^x \leftarrow \min(\boldsymbol{\ell}_j^x, \boldsymbol{x}), \mathbf{u}_j^x \leftarrow \max(\mathbf{u}_j^x, \boldsymbol{x})$ ▷ *update extent of node* j
    if $j \notin$ leaves(T) **then** ▷ *return if* j *is a leaf node, else recurse down the tree*
      if $x_{\delta_j} \leq \xi_j$ **then** child$(j) =$ left(j) **else** child$(j) =$ right(j)
      ExtendMondrianBlock$(\mathcal{T}, \lambda,$ child$(j), \mathcal{D})$ ▷ *recurse on child containing* x

deep trees due to the high cost associated with maintaining candidate quality scores for the fringe of potential children (Denil et al., 2013).

There has been some work on incremental induction of decision trees, e.g. incremental CART (Crawford, 1989), ITI (Utgoff, 1989), VFDT (Domingos and Hulten, 2000) and dynamic trees (Taddy et al., 2011), but to the best of our knowledge, these are focused on learning decision trees and have not been generalized to online random forests. We do not compare MF to incremental decision trees, since random forests are known to outperform single decision trees.

Bayesian models of decision trees (Chipman et al., 1998; Denison et al., 1998) typically specify a distribution over decision trees; such distributions usually depend on $\boldsymbol{X}$ and lack the projectivity property of the Mondrian process. More importantly, MF performs ensemble model combination and not Bayesian model averaging over decision trees. (See

(Dietterich, 2000) for a discussion on the advantages of ensembles over single models, and (Minka, 2000) for a comparison of Bayesian model averaging and model combination.)

5.7 Empirical evaluation

The purpose of these experiments is to evaluate the predictive performance (test accuracy) of MF as a function of (i) fraction of training data and (ii) training time. We divide the training data into 100 mini-batches and we compare the performance of online random forests (MF, ORF-Saffari (Saffari et al., 2009)) to batch random forests (Breiman-RF, ERT-k, ERT-1) which are trained on the same fraction of the training data. (We compare MF to dynamic trees as well; see Section 5.7.3 for more details.) Our scripts are implemented in Python.[6] We implemented the ORF-Saffari algorithm as well as ERT in Python for timing comparisons. The scripts can be downloaded from the authors' webpages. We did not implement the ORF-Denil (Denil et al., 2013) algorithm since the predictive performance reported in (Denil et al., 2013) is very similar to that of ORF-Saffari and the computational complexity of the ORF-Denil algorithm is worse than that of ORF-Saffari. We used the Breiman-RF implementation in *scikit-learn* (Pedregosa et al., 2011).[7]

We evaluate on four of the five datasets used in (Saffari et al., 2009) — we excluded the *mushroom* dataset as even very simple logical rules achieve $> 99\%$ accuracy on this dataset.[8] We re-scaled the datasets such that each feature takes on values in the range $[0, 1]$ (by subtracting the min value along that dimension and dividing by the range along that dimension, where $\mathsf{range} = \mathsf{max} - \mathsf{min}$).

As is common in the random forest literature (Breiman, 2001), we set the number of trees $M = 100$. For Mondrian forests, we set the lifetime $\lambda = \infty$ and the HNSP discount parameter $\gamma = 10D$. For ORF-Saffari, we set $\mathsf{num\_epochs} = 20$ (number of passes through the training data) and set the other hyper parameters to the values used in (Saffari et al., 2009). For Breiman-RF and ERT, the hyper parameters are set to default values. We repeat each algorithm with five random initializations and report the mean performance. The results are shown in Figure 5.3. (The * in Breiman-RF* indicates *scikit-learn* implementation.)

Comparing test accuracy vs fraction of training data on *usps*, *satimages* and *letter* datasets, we observe that **MF achieves accuracy very close to the batch RF versions** (Breiman-RF, ERT-k, ERT-1) trained on the same fraction of the data. **MF significantly outperforms ORF-Saffari trained on the same fraction of**

[6] http://www.gatsby.ucl.ac.uk/~balaji/mondrianforest/

[7] The *scikit-learn* implementation uses highly optimized C code, hence we do not compare our runtimes with the *scikit-learn* implementation. The ERT implementation in *scikit-learn* achieves very similar test accuracy as our ERT implementation, hence we do not report those results here.

[8] https://archive.ics.uci.edu/ml/machine-learning-databases/mushroom/agaricus-lepiota.names

training data. In batch RF versions, the same training data can be used to evaluate candidate splits at a node and its children. However, in the online RF versions (ORF-Saffari and ORF-Denil), incoming training examples are used to evaluate candidate splits just at a current leaf node and new training data are required to evaluate candidate splits every time a new leaf node is created. Saffari et al. (2009) recommend multiple passes through the training data to increase the effective number of training samples. In a realistic streaming data setup, where training examples cannot be stored for multiple passes, MF would require significantly fewer examples than ORF-Saffari to achieve the same accuracy.

Comparing test accuracy vs training time on *usps*, *satimages* and *letter* datasets, we observe that **MF is at least an order of magnitude faster than re-trained batch versions and ORF-Saffari**. For ORF-Saffari, we plot test accuracy at the end of every additional pass; hence it contains additional markers compared to the top row which plots results after a single pass. Re-training batch RF using 100 mini-batches is unfair to MF; in a streaming data setup where the model is updated when a new training instance arrives, MF would be significantly faster than the re-trained batch versions. Assuming trees are balanced after adding each data point, it can be shown that computational cost of MF scales as $\mathcal{O}(N \log N)$ whereas that of re-trained batch RF scales as $\mathcal{O}(N^2 \log N)$ (Section 5.7.1). Section 5.7.2 shows that the average depth of the forests trained on above datasets scales as $\mathcal{O}(\log N)$.

It is remarkable that choosing splits independent of labels achieves competitive classification performance. This phenomenon has been observed by others as well—for example, Cutler and Zhao (2001) demonstrate that their PERT classifier (which is similar to batch version of MF) achieves test accuracy comparable to Breiman-RF on many real world datasets. However, in the presence of irrelevant features, methods which choose splits independent of labels (MF, ERT-1) perform worse than Breiman-RF and ERT-k (but still better than ORF-Saffari) as indicated by the results on the *dna* dataset. We trained MF and ERT-1 using just the most relevant 60 attributes amongst the 180 attributes[9]—these results are indicated as MF† and ERT-1† in Figure 5.3. We observe that, as expected, filtering out irrelevant features significantly improves performance of MF and ERT-1.

5.7.1 Computational complexity

We discuss the computational complexity associated with a single Mondrian tree. The complexity of a forest is simply M times that of a single tree; however, this computation can be trivially parallelized since there is no interaction between the trees. Assume that the N data points are processed one by one. Assuming the data points form a

[9]See the data description `https://www.sgi.com/tech/mlc/db/DNA.names` for the list of most relevant 60 attributes.

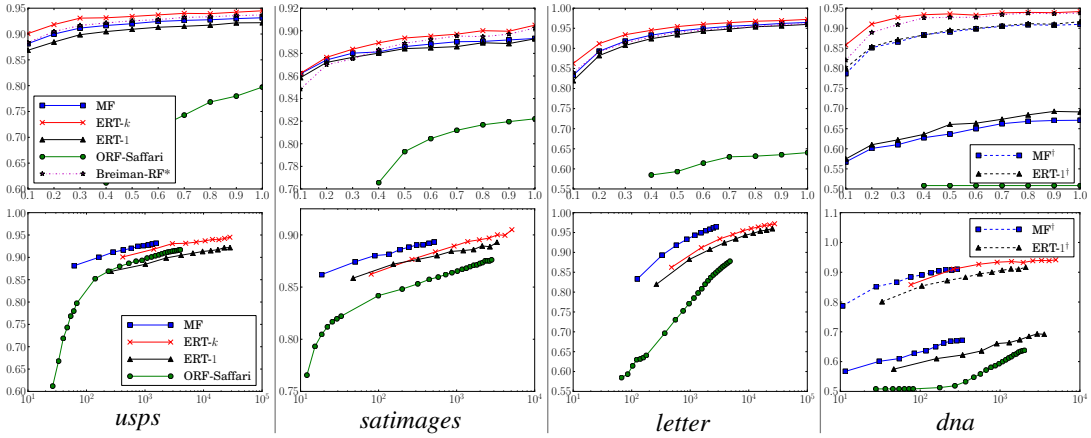


Figure 5.3: Results on various datasets: y-axis is test accuracy in both rows. x-axis is fraction of training data for the top row and training time (in seconds) for the bottom row. We used the pre-defined train/test split. For *usps* dataset $D = 256, K = 10, N_{\text{train}} = 7291, N_{\text{test}} = 2007$; for *satimages* dataset $D = 36, K = 6, N_{\text{train}} = 3104, N_{\text{test}} = 2000$; *letter* dataset $D = 16, K = 26, N_{\text{train}} = 15000, N_{\text{test}} = 5000$; for *dna* dataset $D = 180, K = 3, N_{\text{train}} = 1400, N_{\text{test}} = 1186$.

balanced binary tree after each update, the computational cost of processing the n^{th} data point is at most $\mathcal{O}(\log n)$ (add the data point into its own leaf, update posterior counts for HNSP in bottom-up pass from leaf to root). The overall cost to process N data points is $\mathcal{O}(\sum_{n=1}^{N} \log n) = \mathcal{O}(\log N!)$, which for large N tends to $\mathcal{O}(N \log N)$ (using Stirling approximation for the factorial function). For offline RF and ERT, the expected complexity with n data points is $\mathcal{O}(n \log n)$. The complexity of the re-trained version is $\mathcal{O}(\sum_{n=1}^{N} n \log n) = \mathcal{O}(\log \prod_{n=1}^{N} n^n)$, which for large N tends to $\mathcal{O}(N^2 \log N)$ (using asymptotic expansion of the hyper factorial function).

5.7.2 Depth of trees

We computed the average depth of the trees in the forest, where depth of a leaf node is weighted by fraction of data points at that leaf node. The hyper-parameter settings and experimental setup are described in Section 5.7. Table 5.1 reports the average depth (and standard deviations) for Mondrian forests trained on different datasets. The values suggest that the depth of the forest scales as $\log N$ rather than N.

Dataset	N_{train}	$\log_2 N_{\text{train}}$	depth
usps	7291	12.8	19.1 ± 1.3
satimages	3104	11.6	17.4 ± 1.6
letter	15000	13.9	23.2 ± 1.8
dna	1400	10.5	12.0 ± 0.3

Table 5.1: Average depth of Mondrian forests trained on different datasets.

5.7.3 Comparison to dynamic trees

Dynamic trees (Taddy et al., 2011) approximate the Bayesian posterior over decision trees in an online fashion. Specifically, dynamic trees maintain a particle approximation to the true posterior; the prediction at a test point is a weighted average of the predictions made by the individual particles. While this averaging procedure appears similar to online random forests at first sight, there is a key difference: MF (and other random forests) performs ensemble model combination whereas dynamic trees use Bayesian model averaging. In the limit of infinite data, the Bayesian posterior would converge to a single tree (Minka, 2000), whereas MF would still average predictions over multiple trees. Hence, we expect MF to outperform dynamic trees in scenarios where a single decision tree is insufficient to explain the data.

To experimentally validate our hypothesis, we evaluate the empirical performance of dynamic trees using the dynaTree[10] R package provided by the authors of the paper. Note that while dynamic trees can use 'linear leaves' (strong since prediction at a leaf depends on X) or 'constant leaves' for regression tasks, they use 'multinomial leaves' for classification tasks which corresponds to a 'weak learner'. We set the number of particles to 100 (equals the number of trees used in MF) and the number of passes, $R = 2$ (their code does not support $R = 1$) and set the remaining parameters to their default values. Fig. 5.4 compares the performance of dynamic trees to MF and other random forest variants. (The performance of all methods other than dynamic trees is identical to that of Fig. 5.3.)

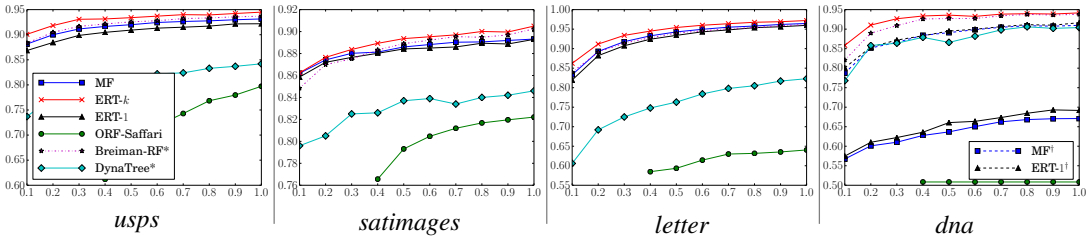


Figure 5.4: Results on various datasets: y-axis is test accuracy in both rows. x-axis is fraction of training data. The setup is identical to that of Fig. 5.3. MF achieves significantly higher test accuracies than dynamic trees on *usps*, *satimages* and *letter* datasets and MF† achieves similar test accuracy as dynamic trees on the *dna* dataset.

We observe that MF achieves significantly higher test accuracies than dynamic trees on *usps*, *satimages* and *letter* datasets. On *dna* dataset, dynamic trees outperform MF (indicating the usefulness of using labels to guide splits) — however, MF with feature selection (MF†) achieves similar performance as dynamic trees. All the batch random forest methods are superior to dynamic trees which suggests that decision trees are not sufficient to explain these real world datasets and that model combination is helpful.

[10] http://cran.r-project.org/web/packages/dynaTree/index.html

5.8 Discussion

We have introduced *Mondrian forests*, a novel class of random forests, which can be trained incrementally in an efficient manner. MF significantly outperforms existing online random forests in terms of training time as well as number of training instances required to achieve a particular test accuracy. Remarkably, MF achieves competitive test accuracy to batch random forests trained on the same fraction of the data. MF is unable to handle lots of irrelevant features (since splits are chosen independent of the labels)—one way to use labels to guide splits is via the Sequential Monte Carlo algorithm for decision trees described in Chapter 3. The computational complexity of MF is linear in the number of dimensions (since rectangles are represented explicitly) which could be expensive for high dimensional data; we will address this limitation in future work. Random forests have been tremendously influential in machine learning for a variety of tasks; hence lots of other interesting extensions of this work are possible, e.g. MF for regression (see Chapter 6), theoretical bias-variance analysis of MF, and extensions of MF that use hyperplane splits instead of axis-aligned splits.

Chapter 6

Mondrian forests for regression

6.1 Introduction

Gaussian process (GP) regression is popular due to its ability to deliver both accurate non-parametric predictions and estimates of uncertainty for those predictions. The dominance of GP regression in applications such as Bayesian optimization, where uncertainty estimates are key to balance exploration and exploitation, is a testament to the quality of GP uncertainty estimates.

Unfortunately, the computational cost of GPs is cubic in the number of data points, making them computationally very expensive for large scale non-parametric regression tasks. (Specifically, we focus on the scenario where the number of data points N is large, but the number of dimensions D is modest.) Steady progress has been made over the past decade on scaling GP inference to big data, including some impressive recent work such as (Deisenroth and Ng, 2015; Gal et al., 2014; Hensman et al., 2013).

Ensembles of randomized decision trees, also known as *decision forests*, are popular for (non-probabilistic) non-parametric regression tasks, often achieving state-of-the-art predictive performance (Caruana and Niculescu-Mizil, 2006). The most popular decision forest variants are *random forests* (Breiman-RF) introduced by Breiman (2001) and *extremely randomized trees* (ERT) introduced by Geurts et al. (2006). The computational cost of learning decision forests is typically $\mathcal{O}(N \log N)$ and the computation across the trees in the forest can be parallelized trivially, making them attractive for large scale regression tasks. While decision forests usually yield good predictions (as measured by, e.g., mean squared error or classification accuracy), the uncertainty estimates of decision forests are not as good as those produced by GPs. For instance, Jitkrittum et al. (2015) compare the uncertainty estimates of decision forests and GPs on a simple regression problem where the test distributions are different from the training distribution. As we move away from the training distribution, GP predictions smoothly return to the prior and exhibit higher uncertainty. However, the uncertainty estimates of decision forests

are less smooth and do not exhibit this desirable property.

Our goal is to combine the desirable properties of GPs (good uncertainty estimates, probabilistic setup) with those of decision forests (computational speed). To this end, we extend Mondrian forests (MFs), introduced in Chapter 5 for classification tasks, to non-parametric regression tasks. Unlike usual decision forests, we use a probabilistic model within each tree to model the labels. Specifically, we use a hierarchical Gaussian prior over the leaf node parameters and compute the posterior parameters efficiently using Gaussian belief propagation (Murphy, 2012). Due to special properties of Mondrian processes, their use as the underlying randomization mechanism results in a desirable uncertainty property: the prediction at a test point shrinks to the prior as the test point moves further away from the observed training data points. We demonstrate that, as a result, Mondrian forests yields better uncertainty estimates.

The chapter is organized as follows: in Section 6.2, we briefly review Mondrian forests. We present Mondrian forests for regression in Section 6.3 and discuss inference and prediction in detail. We present experiments in Section 6.5 that demonstrate that (i) Mondrian forests produce better uncertainty estimates than Breiman-RF and ERT when test distribution is different from training distribution, (ii) Mondrian forests outperform or achieve comparable performance to large scale approximate GPs in terms of mean squared error (MSE) or negative log predictive density (NLPD), thus making them well suited for large scale regression tasks where uncertainty estimates are important, and (iii) Mondrian forests outperform (or perform as well as) decision forests on Bayesian optimization tasks, where predictive uncertainty is important (since it guides the exploration-exploitation tradeoff). Finally, we discuss avenues for future work in Section 6.6.

6.2 Mondrian forests

In Chapter 5, we introduced Mondrian forests (MFs) for classification tasks. For completeness, we briefly review Mondrian trees before describing how MFs can be applied to regression. The main difference from Chapter 5 is the stopping criterion. Specifically, in Chapter 5, we set $\lambda = \infty$ and stopped splitting a node if all the class labels of the data points within the node were identical. We follow a similar approach for regression: we do not split a node which has less than min_samples_split number of data points. (See also the discussion in Section 5.5.1.) Since this is a relatively minor modification, readers familiar with Chapter 5 can safely skip the rest of Section 6.2.

Our problem setup is the following: given N labeled examples $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N) \in \mathcal{X} \times \mathbb{R}$ as training data, our task is to predict labels[1] $y \in \mathbb{R}$ for unlabeled test points $\boldsymbol{x} \in \mathcal{X}$ as well as provide corresponding estimates of uncertainty. Let $\boldsymbol{X}_{1:n} := (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$,

[1] We refer to $y \in \mathbb{R}$ as label even though it is common in statistics to refer to $y \in \mathbb{R}$ as response instead of label.

$Y_{1:n} := (y_1, \ldots, y_n)$, and $\mathcal{D}_{1:n} := (\boldsymbol{X}_{1:n}, Y_{1:n})$. We refer to Section 2.2 and Figure 2.1 a review of decision trees and our notation.

6.2.1 Mondrian trees and Mondrian forests

We briefly review Mondrian trees here and refer to Section 5.3 for further details. A Mondrian process (Roy and Teh, 2009) is a continuous-time Markov process $(\mathcal{M}_t : t \geq 0)$, where, for every $t \geq s \geq 0$, $\mathcal{M}_t$ is a hierarchical binary partition of $\mathcal{X}$ and a refinement of $\mathcal{M}_s$. **Mondrian trees** are restrictions of Mondrian processes to a finite set of points. Figure 5.1 illustrates the difference between decision trees and Mondrian trees. In particular, a Mondrian tree T is a tuple $(\mathsf{T}, \boldsymbol{\delta}, \boldsymbol{\xi}, \boldsymbol{\tau})$, where $(\mathsf{T}, \boldsymbol{\delta}, \boldsymbol{\xi})$ is a decision tree and $\boldsymbol{\tau} = \{\tau_j\}_{j \in \mathsf{T}}$ specifies a **split time** $\tau_j \geq 0$ with each node j. Split times increase with depth, i.e., $\tau_j > \tau_{\mathsf{parent}(j)}$ and play an important role in online updates.

The expected depth of a Mondrian tree is parametrized by a non-negative *lifetime* parameter $\lambda > 0$. Since it is difficult to specify λ, in Chapter 5, we set $\lambda = \infty$ and stopped splitting a node if all the class labels of the data points within the node were identical. We follow a similar approach for regression: we do not split a node which has less than $\mathsf{min\_samples\_split}$ number of data points.[2] Given a bound $\mathsf{min\_samples\_split}$ and training data $\mathcal{D}_{1:n}$, the generative process for sampling Mondrian trees is described in Algorithms 6.1 and 6.2.

Algorithm 6.1 $\mathsf{SampleMondrianTree}\big(\mathcal{D}_{1:n}, \mathsf{min\_samples\_split}\big)$

Initialize: $\mathsf{T} = \emptyset$, $\mathsf{leaves}(\mathsf{T}) = \emptyset$, $\boldsymbol{\delta} = \emptyset$, $\boldsymbol{\xi} = \emptyset$, $\boldsymbol{\tau} = \emptyset$, $N_\epsilon = \{1, 2, \ldots, n\}$ ▷ *initialize empty tree*
$\mathsf{SampleMondrianBlock}\big(\epsilon, \mathcal{D}_{N_\epsilon}, \mathsf{min\_samples\_split}\big)$ ▷ *Algorithm 6.2*

Algorithm 6.2 $\mathsf{SampleMondrianBlock}\big(j, \mathcal{D}_{N_j}, \mathsf{min\_samples\_split}\big)$

Add j to T and for all d, set $\ell^x_{jd} = \mathsf{min}(\boldsymbol{X}_{N_j,d}), u^x_{jd} = \mathsf{max}(\boldsymbol{X}_{N_j,d})$ ▷ *dimension-wise* min *and* max
if $|N_j| \geq \mathsf{min\_samples\_split}$ **then** ▷ *j is an internal node. $|N_j|$ denotes # data points.*
    Sample E from exponential distribution with rate $\sum_d (u^x_{jd} - \ell^x_{jd})$
    Set $\tau_j = \tau_{\mathsf{parent}(j)} + E$
    Sample split dimension δ_j, choosing d with probability proportional to $u^x_{jd} - \ell^x_{jd}$
    Sample split location ξ_j uniformly from interval $[\ell^x_{j\delta_j}, u^x_{j\delta_j}]$
    Set $N_{\mathsf{left}(j)} = \{n \in N_j : \boldsymbol{X}_{n,\delta_j} \leq \xi_j\}$ and $N_{\mathsf{right}(j)} = \{n \in N_j : \boldsymbol{X}_{n,\delta_j} > \xi_j\}$
    $\mathsf{SampleMondrianBlock}\big(\mathsf{left}(j), \mathcal{D}_{N_{\mathsf{left}(j)}}, \mathsf{min\_samples\_split}\big)$
    $\mathsf{SampleMondrianBlock}\big(\mathsf{right}(j), \mathcal{D}_{N_{\mathsf{right}(j)}}, \mathsf{min\_samples\_split}\big)$
else ▷ *j is a leaf node*
    Set $\tau_j = \infty$ and add j to $\mathsf{leaves}(\mathsf{T})$

[2] Specifying $\mathsf{min\_samples\_split}$ instead of max-depth is common in decision forests, cf. (Geurts et al., 2006).

The process is similar to top-down induction of decision trees except for the following key differences: (i) splits in a Mondrian tree are committed only within the range of training data (see Figure 5.1), and (ii) the split dimensions and locations are chosen independent of the labels and uniformly within B_j^x (see lines 5, 6 of Algorithm 6.2). A **Mondrian forest** consists of M i.i.d. Mondrian trees $\mathcal{T}_m = (\mathsf{T}_m, \boldsymbol{\delta}_m, \boldsymbol{\xi}_m, \boldsymbol{\tau}_m)$ for $m = 1, \ldots, M$. See Chapter 5 for further details.

Mondrian trees can be updated online in an efficient manner and remarkably, the distribution of trees sampled from the online algorithm is identical to the corresponding batch counterpart. We use the batch version of Mondrian forests (Algorithms 6.1 and 6.2) in all of our experiments except the Bayesian optimization experiment in Section 6.5.3. Since we do not evaluate the computational advantages of online Mondrian forest, using a batch Mondrian forest in the Bayesian optimization experiment would not affect the reported results. For completeness, we describe the online updates in Algorithms 6.3 and 6.4.

Pseudocode for online learning of Mondrian trees

The online updates are shown in Algorithms 6.3 and 6.4. The prediction step is detailed in Algorithm 6.5.

Algorithm 6.3 ExtendMondrianTree($\mathcal{T}, \mathcal{D}$, min_samples_split)

1: Input: Tree $\mathcal{T} = (\mathsf{T}, \boldsymbol{\delta}, \boldsymbol{\xi}, \boldsymbol{\tau})$, new training instance $\mathcal{D} = (\boldsymbol{x}, y)$
2: ExtendMondrianBlock($\mathcal{T}, \epsilon, \mathcal{D}$, min_samples_split) ▷ *Algorithm 6.4*

Algorithm 6.4 ExtendMondrianBlock($\mathcal{T}, j, \mathcal{D}$, min_samples_split)

1: Set $\mathbf{e}^\ell = \max(\boldsymbol{\ell}_j^x - \boldsymbol{x}, 0)$ and $\mathbf{e}^u = \max(\boldsymbol{x} - \mathbf{u}_j^x, 0)$ ▷ $\mathbf{e}^\ell = \mathbf{e}^u = \mathbf{0}_D$ *if* $\boldsymbol{x} \in B_j^x$
2: Sample E from exponential distribution with rate $\sum_d (e_d^\ell + e_d^u)$
3: **if** $\tau_{\mathsf{parent}(j)} + E < \tau_j$ **then** ▷ *introduce new parent for node* j
4: Sample split dimension δ, choosing d with probability proportional to $e_d^\ell + e_d^u$
5: Sample split location ξ uniformly from interval $[u_{j,\delta}^x, x_\delta]$ **if** $x_\delta > u_{j,\delta}^x$ **else** $[x_\delta, \ell_{j,\delta}^x]$.
6: Insert a new node $\tilde{j}$ just above node j in the tree, and a new leaf j'', sibling to j, where
7: $\delta_{\tilde{j}} = \delta$, $\xi_{\tilde{j}} = \xi$, $\tau_{\tilde{j}} = \tau_{\mathsf{parent}(j)} + E$, $\boldsymbol{\ell}_{\tilde{j}}^x = \min(\boldsymbol{\ell}_j^x, \boldsymbol{x})$, $\mathbf{u}_{\tilde{j}}^x = \max(\mathbf{u}_j^x, \boldsymbol{x})$
8: $j'' = \mathsf{left}(\tilde{j})$ **iff** $x_{\delta_{\tilde{j}}} \leq \xi_{\tilde{j}}$
9: SampleMondrianBlock$(j'', \mathcal{D}$, min_samples_split$)$
10: **else**
11: Update $\boldsymbol{\ell}_j^x \leftarrow \min(\boldsymbol{\ell}_j^x, \boldsymbol{x}), \mathbf{u}_j^x \leftarrow \max(\mathbf{u}_j^x, \boldsymbol{x})$ ▷ *update extent of node* j
12: **if** $j \notin \mathsf{leaves}(\mathsf{T})$ **then** ▷ *return if* j *is a leaf node, else recurse down the tree*
13: **if** $x_{\delta_j} \leq \xi_j$ **then** $\mathsf{child}(j) = \mathsf{left}(j)$ **else** $\mathsf{child}(j) = \mathsf{right}(j)$
14: ExtendMondrianBlock$(\mathcal{T}, \mathsf{child}(j), \mathcal{D}$, min_samples_split$)$ ▷ *recurse on child containing* $\mathcal{D}$

6.3 Model, hierarchical prior, and predictive posterior for labels

In this section, we describe a probabilistic model that will determine the predictive label distribution, $p_{\mathcal{T}}(y|\boldsymbol{x}, \mathcal{D}_{1:N})$, for a tree $\mathcal{T} = (\mathsf{T}, \boldsymbol{\delta}, \boldsymbol{\xi}, \boldsymbol{\tau})$, dataset $\mathcal{D}_{1:N}$, and test point $\boldsymbol{x}$. Let $\mathsf{leaf}(\boldsymbol{x})$ denote the unique leaf node $j \in \mathsf{leaves}(\mathsf{T})$ such that $\boldsymbol{x} \in B_j$. Like with Mondrian forests for classification, we want the predictive label distribution at $\boldsymbol{x}$ to be a smoothed version of the empirical distribution of labels for points in $B_{\mathsf{leaf}(\boldsymbol{x})}$ and in $B_{j'}$ for nearby nodes j'. We will also achieve this smoothing via a hierarchical Bayesian approach: every node is associated with a label distribution, and a prior is chosen under which the label distribution of a node is similar to that of its parent's. The predictive $p_{\mathcal{T}}(y|\boldsymbol{x}, \mathcal{D}_{1:N})$ is then obtained via marginalization.

As is common in the decision tree literature, we assume the labels within each block are independent of $\boldsymbol{X}$ given the tree structure. In Chapter 5, used a hierarchy of normalized stable processes (HNSP) prior for classification problems. In this chapter, we focus on the case of real-valued labels. Let $\mathcal{N}(\mu, v)$ denote a Gaussian distribution with mean μ and variance v. For every $j \in \mathsf{T}$, let μ_j be a mean parameter (of a Gaussian distribution over the labels) at node j, and let $\boldsymbol{\mu} = \{\mu_j : j \in \mathsf{T}\}$. We assume the labels within a leaf node are Gaussian distributed:

$$y_n|\mathcal{T}, \boldsymbol{\mu} \sim \mathcal{N}(\mu_{\mathsf{leaf}(\boldsymbol{x}_n)}, \sigma_y^2) \tag{6.1}$$

where σ_y^2 is a parameter specifying the variance of the (measurement) noise.

We use the following hierarchical Gaussian prior for $\boldsymbol{\mu}$: For hyperparameters μ_H, γ_1, γ_2, let

$$\mu_\epsilon|\mu_H \sim \mathcal{N}(\mu_H, \phi_\epsilon), \qquad \text{and} \qquad \mu_j|\mu_{\mathsf{parent}(j)} \sim \mathcal{N}(\mu_{\mathsf{parent}(j)}, \phi_j),$$

where $\phi_j = \gamma_1\sigma(\gamma_2\tau_j) - \gamma_1\sigma(\gamma_2\tau_{\mathsf{parent}(j)})$ with the convention that $\tau_{\mathsf{parent}(\epsilon)} = 0$, and $\sigma(t) = (1 + e^{-t})^{-1}$ denotes the sigmoid function.

Before discussing the details of posterior inference, we provide some justification for the details of this model: Recall that τ_j increases as we go down the tree, and so the use of the sigmoid $\sigma(\cdot)$ encodes the prior assumption that children are expected to be more similar to their parents as depth increases. The Gaussian hierarchy is *closed under marginalization*, i.e.,

$$\begin{aligned} \mu_\epsilon|\mu_H &\sim \mathcal{N}(\mu_H, \phi_\epsilon) \\ \mu_0|\mu_\epsilon, \mu_H &\sim \mathcal{N}(\mu_\epsilon, \phi_0) \end{aligned} \qquad \textit{implies} \qquad \mu_0|\mu_H \sim \mathcal{N}(\mu_H, \phi_\epsilon + \phi_0),$$

where $\phi_\epsilon + \phi_0 = \gamma_1\sigma(\gamma_2\tau_\epsilon) - \gamma_1\sigma(\gamma_2 0) + \gamma_1\sigma(\gamma_2\tau_0) - \gamma_1\sigma(\gamma_2\tau_\epsilon) = \gamma_1\sigma(\gamma_2\tau_0) - \gamma_1\sigma(\gamma_2 0)$. Therefore, we can introduce intermediate nodes without changing the predictive dis-

tribution. In Section 6.3.4, we show that a test data point can branch off into its own node: the hierarchical prior is critical for smoothing predictions.

Given training data $\mathcal{D}_{1:N}$, our goal is to compute the posterior density over $\boldsymbol{\mu}$:

$$p_{\mathcal{T}}(\boldsymbol{\mu}|\mathcal{D}_{1:N}) \propto p_{\mathcal{T}}(\boldsymbol{\mu}) \prod_{n=1}^{N} \mathcal{N}(y_n|\mu_{\mathsf{leaf}(\boldsymbol{x}_n)}, \sigma_y^2). \tag{6.2}$$

The posterior over $\boldsymbol{\mu}$ will be used during the prediction step described in Section 6.3.4. Note that the posterior over $\boldsymbol{\mu}$ is computed independently for each tree, and so can be parallelized trivially.

6.3.1 Gaussian belief propagation

We perform posterior inference using belief propagation (Pearl, 1988). Since the prior and likelihood are Gaussian, all the messages can be computed analytically and the posterior over $\boldsymbol{\mu}$ is also Gaussian. Since the hierarchy has a tree structure, the posterior can be computed in time that scales linearly with the number of nodes in the tree, which is typically $\mathcal{O}(N)$, hence posterior inference is efficient compared to non-tree-structured Gaussian processes whose computational cost is typically $\mathcal{O}(N^3)$. Message passing in trees is a well-studied problem, and so we refer the reader to (Murphy, 2012, Chapter 20) for details.

6.3.2 Hyperparameter heuristic

In Chapter 5, we stopped splitting a Mondrian block whenever all the class labels were identical.[3] We adopt a similar strategy here and stop splitting a Mondrian block if the number of samples is fewer than a parameter $\mathsf{min\_samples\_split}$. It is common in decision forests to require a minimum number of samples in each leaf, for instance Breiman (2001) and Geurts et al. (2006) recommend setting $\mathsf{min\_samples\_leaf} = 5$ for regression problems. We set $\mathsf{min\_samples\_split} = 10$.

Next, we describe how we choose the hyperparameters $\boldsymbol{\theta} = \{\mu_H, \gamma_1, \gamma_2, \sigma_y^2\}$. For simplicity, we use the same values of these hyperparameters for all the trees; it is possible to optimize these parameters for each tree independently, and would be interesting to evaluate this extra flexibility empirically. Ideally, one might choose hyperparameters by optimizing the marginal likelihood (computed as a byproduct of belief propagation) by, e.g., gradient descent. We use a simpler approach here: we optimize the *product of label*

[3] Technically, the Mondrian tree is *paused* in the online setting and splitting resumes once a block contains more than one distinct label. However, since we only deal with the batch setting, we stop splitting homogeneous blocks.

marginals, integrating out $\boldsymbol{\mu}$ for each label individually, i.e.,

$$q(Y|\boldsymbol{\theta}, \mathcal{T}) = \prod_{j \in \mathsf{leaves(T)}} \prod_{n \in N_j} \mathcal{N}(y_n | \mu_H, \phi_j - \phi_{\mathsf{parent}(\epsilon)} + \sigma_y^2).$$

Since $\tau_j = \infty$ at the leaf nodes, we have

$$\begin{aligned} \phi_j - \phi_{\mathsf{parent}(\epsilon)} &= \gamma_1 \sigma(\gamma_2 \tau_j) - \gamma_1 \sigma(\gamma_2 0) \\ &= \gamma_1 (\sigma(\infty) - \sigma(0)) \\ &= \frac{\gamma_1}{2}. \end{aligned}$$

If the noise variance is known, σ_y^2 can be set to the appropriate value. In our case, the noise variance is unknown; hence, we parametrize σ_y^2 as γ_1/K and set $K = \mathsf{min}(2000, 2N)$ to ensure that the noise variance σ_y^2 is a non-zero fraction of the total variance $\gamma_1/2 + \gamma_1/K$.

We maximize $q(Y|\boldsymbol{\theta}, \mathcal{T})$ over μ_H, γ_1, and K, leading to

$$\begin{aligned} \mu_H &= \frac{1}{N} \sum_n y_n, \\ \gamma_1 (\frac{1}{2} + \frac{1}{K}) &= \frac{1}{N} \sum_n (y_n - \mu_H)^2. \end{aligned}$$

Note that we could have instead performed gradient descent on the actual marginal likelihood produced as a byproduct of belief propagation. It would be interesting to investigate this.

The likelihood $q(Y|\boldsymbol{\theta}, \mathcal{T})$ does not depend on γ_2, and so we cannot choose γ_2 by optimizing it. We know, however, that τ increases with N. Moreover, in Section 5.7.1, we observed that the average tree depths were 2-3 times $\log_2(N)$ in practice. We therefore pre-process the training data to lie in $[0, 1]^D$ and set $\gamma_2 = \frac{D}{20 \log_2 N}$ since (i) τ increases with tree depth and the tree depth is $\mathcal{O}(\log_2 N)$ assuming balanced trees and (ii) τ is inversely proportional to D. In Section 6.3.3, we describe a fast approximation which does not involve estimation of γ_1, γ_2.

6.3.3 Fast approximation to message passing and hyperparameter estimation

So far, we have focused on batch learning setting. Another advantage of Mondrian forests is that the trees can be efficiently updated online with computational complexity $\sum_{n=1}^{N} \mathcal{O}(\log n) = \mathcal{O}(N \log N)$. Note that the cost of updating the Mondrian tree structure is $\mathcal{O}(\log n)$, however exact message passing and hyperparameter estimation cost $\mathcal{O}(n)$ (since addition of a single point affects the predictive posterior at all the nodes). To maintain the $\mathcal{O}(\log n)$ cost, we suggest a fast $\mathcal{O}(\log n)$ approximation to

exact message passing which costs $\mathcal{O}(n)$. We use these fast online updates in our Bayesian optimization experiments in Section 6.5.3.

Under this approximation, the Gaussian posterior at each node is approximated by a Gaussian distribution whose mean and variance are given by the empirical mean and variance of the data points at that node. This approximation is better suited for online applications since adding a new data point involves just updating mean and variance for all the nodes along the path from root to a leaf, hence the overall cost is linear in the depth of the tree. Another advantage of this approximation is that we only need to set the noise variance σ_y^2 and do not need to set the hyper-parameters $\{\mu_H, \gamma_1, \gamma_2\}$.

Since our initial publication, we have learnt that this Gaussian posterior approximation is similar to a random forest modification independently proposed in Hutter et al. (2014, §4.3.2). In (Hutter et al., 2014), each tree outputs a predictive mean and variance equal to the empirical mean and variance of the labels at the leaf node of the decision tree. However, there is an additional level of smoothing in MFs that is not present in (Hutter et al., 2014). Specifically, the prediction from a Mondrian tree, described in (6.3), is a weighted mixture of predictions from nodes along the path from the root to the leaf. Moreover, the weights account for the distance between the test point from the training data, thereby ensuring that the predictions shrink to the prior as we move farther away from the training data.

6.3.4 Predictive variance computation

The prediction step in a Mondrian regression tree is similar to that in a Mondrian classification tree described in Section 5.5.3, except that at each node of the tree, we predict a Gaussian posterior over y rather than predict a posterior over class probabilities. Recall that a prediction from a vanilla decision tree is just the average of the training labels in $\mathsf{leaf}(\boldsymbol{x})$. Unlike decision trees, in a Mondrian tree, a test point $\boldsymbol{x}$ *can potentially 'branch off' the existing Mondrian tree at any point along the path from root to* $\mathsf{leaf}(\boldsymbol{x})$. Hence, the predictive posterior over y from a given tree $\mathcal{T}$ is a mixture of Gaussians of the form

$$p_{\mathcal{T}}(y|\boldsymbol{x}, \mathcal{D}_{1:N}) = \sum_{j \in \mathsf{path}(\mathsf{leaf}(\boldsymbol{x}))} w_j \mathcal{N}(y|m_j, v_j), \tag{6.3}$$

where w_j denotes the weight of each component, given by the probability of branching off just before reaching node j, and m_j, v_j respectively denote the predictive mean and variance.[4] The probability of branching off increases as the test point moves further away from the training data at that particular node; hence, the predictions of MFs exhibit higher uncertainty as we move farther from the training data. For completeness,

[4]Strictly speaking, we need another component to include the possibility of data point lying within the extent of the leaf node, however we ignore this for simplicity.

we provide pseudocode for computing (6.3) in Algorithm 6.5.

If a test data point branches off to create a new node, the predictive mean at that node is the posterior of the parent of the new node; if we did not have a hierarchy and instead assumed the predictions at leaves were i.i.d, then branching would result in a prediction from the prior, which would lead to suboptimal predictions in most applications. The predictive mean and variance for the mixture of Gaussians are

$$\begin{aligned} \mathbb{E}_{\mathcal{T}}[y] &= \sum_j w_j m_j \qquad \text{and} \\ \mathrm{Var}_{\mathcal{T}}[y] &= \sum_j w_j (v_j + m_j^2) - \big(\mathbb{E}_{\mathcal{T}}[y]\big)^2, \end{aligned} \tag{6.4}$$

and the prediction of the ensemble is then

$$p(y|\boldsymbol{x}, \mathcal{D}_{1:N}) = \frac{1}{M} \sum_m p_{\mathcal{T}_m}(y|\boldsymbol{x}, \mathcal{D}_{1:N}). \tag{6.5}$$

The prediction of the ensemble can be thought of as being drawn from a mixture model over M trees where the trees are weighted equally. The predictive mean and variance of the ensemble can be computed using the formula for mixture of Gaussians (similar to (6.4)). Similar strategy has been used in (Criminisi et al., 2012; Hutter et al., 2014) as well.

Algorithm 6.5 $\mathsf{Predict}\big(\mathcal{T}, \boldsymbol{x}\big)$ (prediction using Mondrian regression tree)

▷ *Description of prediction using a Mondrian tree given by (6.3).*
▷ *The predictive mean, predictive variance and NLPD computation are not shown, but they can be computed easily during the top-down pass using the weights w_j and posterior moments m_j, v_j at node j.*
Initialize $j = \epsilon$ and $p_{\mathsf{NotSeparatedYet}} = 1$
while True **do**
    Set $\Delta_j = \tau_j - \tau_{\mathsf{parent}(j)}$ and $\eta_j(\boldsymbol{x}) = \sum_d \big(\mathsf{max}(x_d - u^x_{jd}, 0) + \mathsf{max}(\ell^x_{jd} - x_d, 0)\big)$
    Set $p^s_j(\boldsymbol{x}) = 1 - \exp\big(-\Delta_j \eta_j(\boldsymbol{x})\big)$
    if $p^s_j(\boldsymbol{x}) > 0$ **then**
        $w_j = p_{\mathsf{NotSeparatedYet}}\, p^s_j(\boldsymbol{x})$
    if $j \in \mathsf{leaves(T)}$ **then**
        $w_j = p_{\mathsf{NotSeparatedYet}}(1 - p^s_j(\boldsymbol{x}))$
        return
    else
        $p_{\mathsf{NotSeparatedYet}} \leftarrow p_{\mathsf{NotSeparatedYet}}(1 - p^s_j(\boldsymbol{x}))$
        if $x_{\delta_j} \le \xi_j$ **then** $j \leftarrow \mathsf{left}(j)$ **else** $j \leftarrow \mathsf{right}(j)$ ▷ *recurse to child where $\boldsymbol{x}$ lies*

6.4 Related work

The work on large scale Gaussian processes can be broadly split into approaches that optimize inducing variables using variational approximations and approaches that distribute computation by using experts that operate on subsets of the data. We refer to (Deisenroth and Ng, 2015) for a recent summary of large scale Gaussian processes. Hensman et al. (2013) and Gal et al. (2014) use stochastic variational inference to speed up GPs, building on the variational bound developed by Titsias (2009). Deisenroth and Ng (2015) present the robust Bayesian committee machine (rBCM), which combines predictions from experts that operate on subsets of data.

Hutter (2009) investigated the use of Breiman-RF for Bayesian optimization and used the empirical variance between trees in the forest as a measure of uncertainty. (Hutter et al. (2014) proposed a further modification, see Section 6.3.3.) Eslami et al. (2014) used a non-standard decision forest implementation where a quadratic regressor is fit at each leaf node, rather than a constant regressor as in popular decision forest implementations. Their uncertainty measure—a sum of the Kullback-Leibler (KL) divergence—is highly specific to their application of accelerating expectation propagation, and so it seems their method is unlikely to be immediately applicable to general non-parametric regression tasks. Indeed, Jitkrittum et al. (2015) demonstrate that the uncertainty estimates proposed by (Eslami et al., 2014) are not as good as kernel methods in their application domain when the test distribution is different from the training distribution. As originally defined, Mondrian forests produce uncertainty estimates for categorical labels, but in Chapter 5, we evaluated the performance on (online) prediction (classification accuracy) without any assessment of the uncertainty estimates.

6.5 Experiments

6.5.1 Comparison of uncertainty estimates of MFs to decision forests

In this experiment, we compare uncertainty estimates of MFs to those of popular decision forests. The prediction of MFs is given by (6.5), from which we can compute the predictive mean and predictive variance.[5] For decision forests, we compute the predictive mean as the average of the predictions from the individual trees and, following Hutter (2009, §11.1.3), compute the predictive variance as the variance of the predictions from the individual trees. We use 25 trees and set $\mathsf{min\_samples\_leaf} = 5$ for decision forests to make them comparable to MFs with $\mathsf{min\_samples\_split} = 10$. We used the ERT and Breiman-RF implementation in *scikit-learn* (Pedregosa et al., 2011) and set the remaining hyperparameters to their default values.

[5] Code available at `http://www.gatsby.ucl.ac.uk/~balaji/mondrianforest/`.

We use a simplified version of the dataset described in (Jitkrittum et al., 2015), where the goal is to predict the outgoing message in expectation propagation (EP) from a set of incoming messages. When the predictions are uncertain, the outgoing message will be re-computed (either numerically or using a sampler), hence predictive uncertainty is crucial in this application. Our dataset consists of two-dimensional features (which are derived from the incoming message) and a single target (corresponding to mean of the outgoing message). The scatter plot of the training data features is shown in Fig. 6.1(a). We evaluate predictive uncertainty on two test distributions, shown in red and blue in Fig. 6.1(a), which contain data points in unobserved regions of the training data.

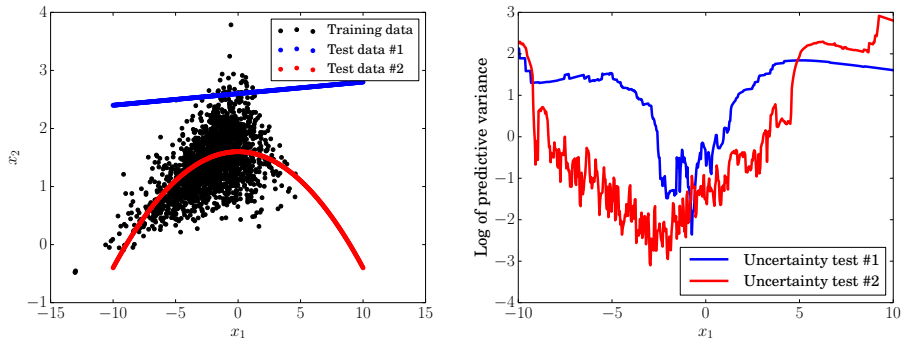
The mean squared error of all the methods are comparable, so we focus just on the predictive uncertainty. Figures 6.1(b), 6.1(c), and 6.1(d) display the predictive uncertainty of MF, ERT and Breiman-RF as a function of x_1. We notice that Breiman-RF's predictions are over-confident compared to MF and ERT. The predictive variance is quite low even in regions where training data has not been observed. The predictive variance of MF is low in regions where training data has been observed ($-5 < x_1 < 5$) and goes up smoothly as we move farther away from the training data; the red test dataset is more similar to the training data than the blue test data and the predictive uncertainty of MF on the blue dataset is higher than that of the red dataset, as one would expect. ERT is less overconfident than Breiman-RF, however its predictive uncertainty is less smooth compared to that of MF.

6.5.2 Comparison to GPs and decision forests on flight delay dataset

In this experiment, we compare decision forest variants to large scale Gaussian processes. Deisenroth and Ng (2015) evaluated a variety of large scale Gaussian processes on the *flight delay* dataset, processed by Hensman et al. (2013), and demonstrate that their method achieves state-of-the-art predictive performance; we evaluate decision forests on the same dataset so that our predictive performance can be directly compared to large scale GPs. The goal is to predict the flight delay from eight attributes, namely, the age of the aircraft (number of years since deployment), distance that needs to be covered, airtime, departure time, arrival time, day of the week, day of the month and month.

Deisenroth and Ng (2015) employed the following strategy: train using the first N data points and use the following 100,000 as test data points. Deisenroth and Ng (2015) created three datasets, setting N to $700K$, $2M$ (million) and $5M$ respectively. We use the same data splits and train MF, Breiman-RF, ERT on these datasets so that our results are directly comparable.[6] We used 10 trees for each forest to reduce the computational burden.

[6] Gal et al. (2014) report the performance of Breiman-RF on these datasets, but they restricted the maximum depth of the trees to 2, which hurts the performance of Breiman-RF significantly. They also use a random train/test split, hence our results are not directly comparable to theirs due to the non-stationarity in the dataset.



(a) Distribution of train/test inputs (labels not depicted)

(b) MF uncertainty

(c) ERT uncertainty

(d) Breiman-RF uncertainty

Figure 6.1: (a) Scatter distribution of training distribution and test distributions. (b-d) Typical uncertainty estimates from a single run of MF, ERT-k and Breiman-RF, respectively, as a function of x_1. (Averaging over multiple runs would create smooth curves while obscuring interesting internal structure to the estimates which an application would potentially suffer from.) As desired, MF becomes less certain away from training inputs, while the other methods report high confidence spuriously.

	700K/100K		2M/100K		5M/100K	
	RMSE	NLPD	RMSE	NLPD	RMSE	NLPD
SVI-GP	33.0	-	-	-	-	-
Dist-VGP	33.0	-	-	-	-	-
rBCM	27.1	9.1	34.4	8.4	**35.5**	8.8
RF	**24.07 ± 0.02**	5.06 ± 0.02*	**27.3 ± 0.01**	5.19 ± 0.02*	39.47 ± 0.02	**6.90 ± 0.05***
ERT	24.32 ± 0.02	6.22 ± 0.03*	27.95 ± 0.02	6.16 ± 0.01*	38.38 ± 0.02	8.41 ± 0.09*
MF	26.57 ± 0.04	**4.89 ± 0.02**	29.46 ± 0.02	**4.97 ± 0.01**	40.13 ± 0.05	**6.91 ± 0.06**

Table 6.1: Comparison of MFs to popular decision forests and large scale GPs on the flight delay dataset. We report average results over 3 runs (with random initializations), along with standard errors. MF achieves significantly better NLPD than rBCM. RF and ERT do not offer a principled way to compute NLPD, hence they are marked with an asterix. Note that SVI-GP, Dist-VGP and rBCM were taken from Deisenroth and Ng (2015).

We evaluate performance by measuring the root mean squared error (RMSE) and negative log predictive density (NLPD). NLPD, defined as the negative logarithm of (6.5), is a popular measure for measuring predictive uncertainty (cf. (Quinonero-Candela et al., 2006, section 4.2)). NLPD penalizes over-confident as well as under-confident predictions since it not only accounts for predictive mean but also the predictive variance. RF and ERT do not offer a principled way to compute NLPD. But, as a simple approximation, we computed NLPD for RF and ERT assuming a Gaussian distribution with mean equal to the average of trees' predictions, variance equal to the variance of trees' predictions.

Table 6.1 presents the results. The RMSE and NLPD results for SVI-GP, Dist-VGP and rBCM results were taken from (Deisenroth and Ng, 2015), who report a standard error lower than 0.3 for all of their results. Table 1 in (Deisenroth and Ng, 2015) shows that rBCM achieves significantly better performance than other large scale GP approximations; hence we only report the performance of rBCM here. It is important to note that the dataset exhibits non-stationarity: as a result, the performance of decision forests as well as GPs is worse on the larger datasets. (This phenomenon was observed by Gal et al. (2014) and Deisenroth and Ng (2015) as well.) On the $700K$ and $2M$ dataset, we observe that decision forests achieve significantly lower RMSE than rBCM. MF achieves significantly lower NLPD compared to rBCM, which suggests that its uncertainty estimates are useful for large scale regression tasks. However, all the decision forests, including MFs, achieve poorer RMSE performance than rBCM on the larger $5M$ dataset. We believe that this is due to the non-stationary nature of the data. To test this hypothesis, we shuffled the 5,100,000 data points to create three new training (test) data sets with $5M$ ($100K$) points; all the decision forests achieved a RMSE in the range 31-34 on these shuffled datasets.

MF outperforms rBCM in terms of NLPD on all three datasets. On the 5M dataset, the NLPD of Breiman-RF is similar to that of MF, however Breiman-RF's uncertainty is not computed in a principled fashion. As an additional measure of uncertainty, we report *probability calibration measures* (akin to those for binary classification cf. `http://scikit-learn.org/stable/modules/calibration.html`), also known as reliability

Dataset	Method	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
700K	Breiman-RF	-0.02	-0.04	-0.05	-0.06	-0.06	-0.06	-0.05	-0.06	-0.07
700K	ERT	-0.04	-0.07	-0.11	-0.14	-0.16	-0.18	-0.19	-0.19	-0.18
700K	MF	-0.01	-0.02	-0.01	0	0.02	0.03	0.03	0.02	0
2M	Breiman-RF	-0.02	-0.04	-0.05	-0.06	-0.05	-0.04	-0.03	-0.03	-0.04
2M	ERT	-0.04	-0.08	-0.12	-0.15	-0.17	-0.18	-0.19	-0.18	-0.16
2M	MF	-0.02	-0.04	-0.05	-0.05	-0.03	0	0.02	0.03	0.01
5M	Breiman-RF	-0.03	-0.06	-0.08	-0.09	-0.1	-0.1	-0.11	-0.1	-0.1
5M	ERT	-0.04	-0.07	-0.11	-0.14	-0.16	-0.18	-0.19	-0.19	-0.18
5M	MF	-0.02	-0.04	-0.05	-0.06	-0.06	-0.05	-0.05	-0.05	-0.07

Table 6.2: Comparison of MFs to popular decision forests on the flight delay dataset. Each entry denotes the difference between the observed fraction minus the ideal fraction (which is shown at the top of the column). Hence, a value of zero implies perfect calibration, a negative value implies overconfidence and a positive value implies under-confident predictor. MF is better calibrated than Breiman-RF and ERT, which are consistently over-confident.

diagrams (DeGroot and Fienberg, 1983), for MF, Breiman-RF and ERT. First, we compute the $z\%$ (e.g. 90%) prediction interval for each test data point based on Gaussian quantiles using predictive mean and variance. Next, we measure what fraction of test observations fall within this prediction interval. For a well-calibrated regressor, the observed fraction should be close to $z\%$. We compute observed fraction for $z = 10\%$ to $z = 90\%$ in increments of 10. We report observed fraction minus ideal fraction since it is easier to interpret—a value of zero implies perfect calibration, a negative value implies over-confidence (a lot more observations lie outside the prediction interval than expected) and a positive value implies under-confidence. The results are shown in Table 6.2. MF is clearly better calibrated than Breiman-RF and ERT, which seem to be consistently over-confident. Since 5M dataset exhibits non-stationarity, MF appears to be over-confident but still outperforms RF and ERT. Deisenroth and Ng (2015) do not report calibration measures and their code is not available publicly, hence we do not report calibration measures for GPs.

6.5.3 Scalable Bayesian optimization

Next, we showcase the usefulness of MFs in a Bayesian optimization (BayesOpt) task. We briefly review the Bayesian optimization setup for completeness and refer the interested reader to (Brochu et al., 2010; Snoek et al., 2012) for further details. Bayesian optimization deals with the problem of identifying the global maximizer (or minimizer) of an unknown (a.k.a. black-box) objective function which is computationally expensive to evaluate.[7] Our goal is to identify the maximizer in as few evaluations as possible. Bayesian optimization is a model-based sequential search approach to solve this problem. Specifically, given n noisy observations, we fit a *surrogate model* such as a Gaussian process or a decision forest and choose the next location based on an *acquisition function* such as upper confidence bound (UCB) (Srinivas et al., 2010) or expected

[7] For a concrete example, consider the task of optimizing the hyperparameters of a deep neural network to maximize validation accuracy.

improvement (EI) (Mockus et al., 1978). The acquisition function trades off exploration versus exploitation by choosing input locations where the predictive mean from the surrogate model is high (exploitation) or the predictive uncertainty of the surrogate model is high (exploration). Hence, a surrogate model with well-calibrated predictive uncertainty is highly desirable. Moreover, the surrogate model has to be re-fit after every new observation is added; while this is not a significant limitation for a few (e.g. 50) observations and scenarios where the evaluation of the objective function is significantly more expensive than re-fitting the surrogate model, the re-fitting can be computationally expensive if one is interested in scalable Bayesian optimization (Snoek et al., 2015).

Hutter (2009) proposed sequential model-based algorithm configuration (SMAC), which uses Breiman-RF as the surrogate model and the uncertainty between the trees as a heuristic measure of uncertainty.[8] Nickson et al. (2014) discuss a scenario where this heuristic produces misleading uncertainty estimates that hinders exploration. It is worth noting that SMAC uses EI as the acquisition function only 50% of the time and uses random search the remaining 50% of the time (which is likely due to the fact that the heuristic predictive uncertainty can collapse to 0). Moreover, SMAC re-fits the surrogate model by running a batch algorithm; the computational complexity of running the batch version N times is $\sum_{n=1}^{N} \mathcal{O}(n \log n) = \mathcal{O}(N^2 \log N)$ (Section 5.7.1).

MFs are desirable for such an application since they can produce principled uncertainty estimates and can be efficiently updated online with computational complexity $\sum_{n=1}^{N} \mathcal{O}(\log n) = \mathcal{O}(N \log N)$. Note that the cost of updating the Mondrian tree structure is $\mathcal{O}(\log n)$, however exact message passing costs $\mathcal{O}(n)$. To maintain the $\mathcal{O}(\log n)$ cost, we use the fast approximation discussed in Section 6.3.3.

We report results on four Bayesian optimization benchmarks used in (Eggensperger et al., 2013; Snoek et al., 2015), consisting of two synthetic functions namely the Branin and Hartmann functions, and two real-world problems, namely optimizing the hyper-parameters of online latent Dirichlet allocation (LDA) and structured support vector machine (SVM). LDA and SVM datasets consist of 288 and 1400 grid points respectively; we sampled Branin and Hartmann functions at 250,000 grid points (to avoid implementing a separate optimizer for optimizing over the acquisition function). For SVM and LDA, some dimensions of the grid vary on a non-linear scale (e.g. $10^0, 10^{-1}, 10^{-2}$); we log-transformed such dimensions and scaled all dimensions to $[0, 1]$ so that all features are on the same scale. We used 10 trees, set $\mathsf{min\_samples\_split} = 2$ and use UCB as the acquisition function[9] for MFs. We repeat our results 15 times (5 times each with 3 different random grids for Branin and Hartmann) and report mean and standard deviation.

Following Eggensperger et al. (2013), we evaluate a fixed number of evaluations for

[8] Hutter et al. (2014, §4.3.2) proposed a further modification to the variance estimation procedure, where each tree outputs a predictive mean and variance, in the spirit of *quantile regression forests* (Meinshausen, 2006). See Section 6.3.3 for a discussion on how this relates to MFs.

[9] Specifically, we set acquisition function = predictive mean + predictive standard deviation.

Dataset (D, #evals)	Oracle	MF	SMAC (Eggensperger et al., 2013)
Branin (2, 200)	-0.398	-0.400 ± 0.005	-0.655 ± 0.27
Hartmann (6, 200)	3.322	3.247 ± 0.109	2.977 ± 0.11
SVM-grid (3, 100)	-1266.2	-1266.36 ± 0.52	-1269.6 ± 2.9
LDA-grid (3, 50)	-24.1	-24.1 ± 0	-24.1± 0.1

Table 6.3: Results on BayesOpt benchmarks: Oracle reports the maximum value on the grid. MF, SMAC (which uses a variant of Breiman-RF) report the maximum value obtained by the respective methods.

each benchmark and measure the maximum value observed. The results are shown in Table 6.3. The SMAC results (using Breiman-RF) were taken from Table 2 of (Eggensperger et al., 2013). Both MF and SMAC identify the optimum for LDA-grid. SMAC does not identify the optimum for Branin and Hartmann functions. We observe that MF finds maxima very close to the true maximum on the grid, thereby suggesting that better uncertainty estimates are useful for better exploration-exploitation tradeoff. The computational advantage of MFs might not be significant with few evaluations, but we expect MFs to be computationally advantageous with thousands of observations, e.g., applications such as scalable Bayesian optimization (Snoek et al., 2015) and reinforcement learning (Ernst et al., 2005).

6.5.4 Failure modes of our approach

No method is a panacea: here we discuss two failure modes of our approach that would be important to address in future work.

First, we expect GPs to perform better than decision forests on extrapolation tasks; a GP with an appropriate kernel (and well-estimated hyperparameter values) can extrapolate beyond the observed range of training data; however, the predictions of decision forests with *constant* predictors at leaf nodes are confined to the range of minimum and maximum observed y. If extrapolation is desired, we need complex regressors (that are capable of extrapolation) at leaf nodes of the decision forest. However, this will increase the cost of posterior inference.

Second, Mondrian forests choose splits independent of the labels; hence irrelevant features can hurt predictive performance (Section 5.7); in the batch setting, one can apply feature selection to filter or down weight the irrelevant features.

6.6 Discussion

We developed a novel and scalable methodology for regression based on Mondrian forests that provides both good predictive accuracy as well as sensible estimates of uncertainty. These uncertainty estimates are important in a range of application areas

including probabilistic numerics, Bayesian optimization and planning. We demonstrate that Mondrian forests deliver better-calibrated uncertainty estimates than existing decision forests, especially in regions far away from the training data. Using a large-scale regression application on flight delay data, we demonstrate that our proposed regression framework can provide both state-of-the-art RMSE and estimates of uncertainty as compared to recent scalable GP approximations. We demonstrated the usefulness of MFs for Bayesian optimization. Since Mondrian forests deliver good uncertainty estimates and can be trained online efficiently, they seem promising for applications such as Bayesian optimization and reinforcement learning. Mondrian forests are also applicable in other statistical inference and decision making applications.

Chapter 7

Summary and future work

This thesis proposes several computationally efficient tree-based algorithms that produce probabilistic predictions.

In Chapters 3 and 4, we proposed a novel class of Bayesian inference algorithms for decision trees and additive trees, based on the sequential Monte Carlo framework. The proposed algorithms mimic classic top-down tree induction algorithms, but replace greedy optimal split selection with a soft version that resamples splits according to the probability of being optimal. The proposed framework not only achieves better computation-vs-performance tradeoff compared to existing MCMC counterparts, but also sheds light on the relationship to their non-Bayesian counterparts.

Bayesian inference over decision tree structures is computationally challenging for big datasets and hard to adapt to the online learning setting. Furthermore, BMA might be suboptimal if there is model misspecification. In Chapters 5 and 6, we propose *Mondrian forests* which use model combination unlike the previous algorithms which perform BMA. Mondrian forests restrict splits to the bounding box of the data and use hierarchical Bayesian smoothing of the leaf node parameters, both of which leads to better uncertainty estimates compared to existing random forests. Moreover, the distribution of online Mondrian forests is equal to the distribution of batch Mondrian forests, making them well-suited for data-efficient online learning compared to existing online random forests. Mondrian forests are computationally much cheaper than the Bayesian algorithms proposed earlier, making them well-suited for large-scale online learning problems.

Of all the algorithms proposed above, Mondrian forests are the most promising due to their computational efficiency as well as nice theoretical properties. It would be interesting to extend Mondrian forests to problems such as semi-supervised learning (cf. (Jagannathan et al., 2013)), density estimation and outlier detection (cf. (Liu et al., 2008)). Mondrian forests cannot handle lots of irrelevant features and their space complexity scales linearly with the number of dimensions; it would be interesting to

develop extensions that do not suffer from these drawbacks. For instance, the memory requirement can be reduced by using random patches as suggested by Louppe and Geurts (2012). Mondrian forests produce better uncertainty estimates than existing decision forests and are hence promising for applications such as active learning (see (Narr et al., 2016) for a recent paper using Mondrian forests for active learning), scalable Bayesian optimization and reinforcement learning. More generally, Mondrian forests are useful for other statistical decision making applications as well. It would be interesting to study the theoretical properties of Mondrian forests; see Biau and Scornet (2015) for a recent review of theoretical and methodological improvements to random forests. The randomization mechanism of Mondrian forests resembles the randomization scheme of *purely uniform random forests*, studied by (Genuer, 2010). However, the hierarchical smoothing in Mondrian forests presents new theoretical challenges and opportunities for improvement. Mondrian forests focus only on axis-aligned splits; it would be interesting to extend these ideas to non-axis-aligned trees using extensions of the Mondrian process, such as the *Ostomachion process* (Fan et al., 2016) and tools from stochastic geometry such as *iterated stable tessellations* (Schreiber et al., 2013).

Bibliography

C. Anagnostopoulos and R. Gramacy. Dynamic trees for streaming and massive data contexts. *arXiv preprint arXiv:1201.5568*, 2012. (page 26)

C. Andrieu, A. Doucet, and R. Holenstein. Particle Markov chain Monte Carlo methods. *J. R. Stat. Soc. Ser. B Stat. Methodol.*, 72(3):269–342, 2010. (pages 13, 47, 52, and 53)

A. Asuncion and D. J. Newman. UCI machine learning repository. `http://www.ics.uci.edu/~mlearn/MLRepository.html`, 2007. (page 34)

M. Bayes and M. Price. An essay towards solving a problem in the doctrine of chances. by the late Rev. Mr. Bayes, FRS communicated by Mr. Price, in a letter to John Canton, AMFRS. *Philosophical Transactions (1683-1775)*, pages 370–418, 1763. (pages 11 and 18)

G. Biau and E. Scornet. A random forest guided tour. *arXiv preprint arXiv:1511.05741*, 2015. (page 98)

J. Bleich, A. Kapelner, E. I. George, S. T. Jensen, et al. Variable selection for BART: an application to gene regulation. *The Annals of Applied Statistics*, 8(3):1750–1781, 2014. (page 59)

A. Bouchard-Côté, S. Sankararaman, and M. I. Jordan. Phylogenetic inference via sequential Monte Carlo. *Systematic biology*, 61(4):579–593, 2012. (page 29)

L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996. (pages 12, 21, 46, and 72)

L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. (pages 12, 22, 26, 37, 46, 60, 61, 73, 75, 80, and 85)

L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and regression trees*. Chapman & Hall/CRC, 1984. (pages 11, 16, and 26)

E. Brochu, V. M. Cora, and N. De Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010. (page 93)

W. Buntine. Learning classification trees. *Stat. Comput.*, 2:63–73, 1992. (pages 12, 19, and 26)

O. Cappé, S. J. Godsill, and E. Moulines. An overview of existing methods and recent advances in sequential Monte Carlo. *Proc. IEEE*, 95(5):899–924, 2007. (page 31)

R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proc. Int. Conf. Mach. Learn. (ICML)*, 2006. (pages 12, 20, 21, 46, 60, and 80)

H. Chipman and R. E. McCulloch. Hierarchical priors for Bayesian CART shrinkage. *Stat. Comput.*, 10(1):17–24, 2000. (pages 26 and 45)

H. A. Chipman, E. I. George, and R. E. McCulloch. Bayesian CART model search. *J. Am. Stat. Assoc.*, pages 935–948, 1998. (pages 12, 15, 19, 26, 28, 29, 34, 39, 51, and 74)

H. A. Chipman, E. I. George, and R. E. McCulloch. BART: Bayesian additive regression trees. *Ann. Appl. Stat.*, 4(1):266–298, 2010. (pages 12, 20, 21, 24, 44, 46, 47, 48, 49, 50, 51, 52, and 55)

B. Clarke. Comparing Bayes model averaging and stacking when model approximation error cannot be ignored. *J. Mach. Learn. Res. (JMLR)*, 4:683–712, 2003. (page 24)

S. L. Crawford. Extensions to the CART algorithm. *Int. J. Man-Machine Stud.*, 31(2): 197–217, 1989. (page 74)

A. Criminisi, J. Shotton, and E. Konukoglu. Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. *Found. Trends Comput. Graphics and Vision*, 7(2–3):81–227, 2012. (pages 12, 22, 60, 72, and 88)

A. Cutler and G. Zhao. PERT - Perfect Random Tree Ensembles. *Comput. Sci. and Stat.*, 33:490–497, 2001. (pages 73 and 76)

M. H. DeGroot and S. E. Fienberg. The comparison and evaluation of forecasters. *The statistician*, pages 12–22, 1983. (page 93)

M. P. Deisenroth and J. W. Ng. Distributed Gaussian processes. In *Proc. Int. Conf. Mach. Learn. (ICML)*, 2015. (pages 80, 89, 90, 92, and 93)

P. Del Moral, A. Doucet, and A. Jasra. Sequential Monte Carlo samplers. *J. R. Stat. Soc. Ser. B Stat. Methodol.*, 68(3):411–436, 2006. (page 31)

M. Denil, D. Matheson, and N. de Freitas. Consistency of online random forests. In *Proc. Int. Conf. Mach. Learn. (ICML)*, 2013. (pages 72, 73, 74, and 75)

D. G. T. Denison, B. K. Mallick, and A. F. M. Smith. A Bayesian CART algorithm. *Biometrika*, 85(2):363–377, 1998. (pages 12, 19, 26, 59, and 74)

T. G. Dietterich. Ensemble methods in machine learning. In *Multiple classifier systems*, pages 1–15. Springer, 2000. (pages 12, 19, 22, and 75)

P. Domingos. Bayesian averaging of classifiers and the overfitting problem. In *Proc. Int. Conf. Mach. Learn. (ICML)*, 2000. (page 24)

P. Domingos and G. Hulten. Mining high-speed data streams. In *Proc. 6th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. (KDD)*, pages 71–80. ACM, 2000. (page 74)

R. Douc, O. Cappé, and E. Moulines. Comparison of resampling schemes for particle filtering. In *Image Sig. Proc. Anal.*, pages 64–69, 2005. (page 35)

K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, and K. Leyton-Brown. Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice*, 2013. (pages 94 and 95)

D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *J. Mach. Learn. Res. (JMLR)*, 6:503–556, 2005. (page 95)

S. A. Eslami, D. Tarlow, P. Kohli, and J. Winn. Just-in-time learning for fast and flexible inference. In *Adv. Neural Information Proc. Systems (NIPS)*, pages 154–162, 2014. (page 89)

X. Fan, B. Li, Y. Wang, Y. Wang, and F. Chen. The Ostomachion process. In *Proc. AAAI Conf. Artif. Intell*, 2016. (page 98)

M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we need hundreds of classifiers to solve real world classification problems? *J. Mach. Learn. Res. (JMLR)*, 15:3133–3181, 2014. (pages 12 and 22)

Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1): 119–139, 1997. (page 20)

Y. Freund, R. Schapire, and N. Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999. (page 12)

J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Ann. Statist*, 29(5):1189–1232, 2001. (pages 12, 20, 26, and 46)

J. H. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002. (pages 20 and 46)

Y. Gal, M. van der Wilk, and C. Rasmussen. Distributed variational inference in sparse Gaussian process regression and latent variable models. In *Adv. Neural Information Proc. Systems (NIPS)*, pages 3257–3265, 2014. (pages 80, 89, 90, and 92)

R. Genuer. Risk bounds for purely uniformly random forests. *arXiv preprint arXiv:1006.2980*, 2010. (page 98)

P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006. (pages 21, 22, 46, 68, 73, 80, 82, and 85)

Z. Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521 (7553):452–459, 2015. (page 11)

J. T. Goodman. A bit of progress in language modeling. *Comput. Speech Lang.*, 15(4): 403–434, 2001. (pages 65 and 66)

N. J. Gordon, D. J. Salmond, and A. F. M. Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *Radar Sig. Proc., IEE Proc. F*, 140(2):107–113, 1993. (page 32)

T. Hastie, R. Tibshirani, et al. Bayesian backfitting (with comments and a rejoinder by the authors). *Statistical Science*, 15(3):196–223, 2000. (page 47)

J. Hensman, N. Fusi, and N. D. Lawrence. Gaussian processes for big data. In *Conf. Uncertainty Artificial Intelligence (UAI)*, pages 282–290, 2013. (pages 80, 89, and 90)

J. A. Hoeting, D. Madigan, A. E. Raftery, and C. T. Volinsky. Bayesian model averaging: a tutorial. *Statistical science*, pages 382–401, 1999. (pages 12 and 19)

F. Hutter. *Automated configuration of algorithms for solving hard computational problems*. PhD thesis, University of British Columbia, 2009. (pages 89 and 94)

F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, 2014. (pages 87, 88, 89, and 94)

R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991. (page 25)

G. Jagannathan, C. Monteleoni, and K. Pillaipakkamnatt. A semi-supervised learning approach to differential privacy. In *Data Mining Workshops (ICDMW), 2013 IEEE 13th International Conference on*, pages 841–848. IEEE, 2013. (page 97)

W. Jitkrittum, A. Gretton, N. Heess, S. Eslami, B. Lakshminarayanan, D. Sejdinovic, and Z. Szabó. Kernel-based just-in-time learning for passing expectation propagation messages. In *Conf. Uncertainty Artificial Intelligence (UAI)*, 2015. (pages 80, 89, and 90)

R. Johnson and T. Zhang. Learning nonlinear functions using regularized greedy forest. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 36(5):942-954*, 2013. (pages 20 and 58)

M. I. Jordan and R. A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural computation*, 6(2):181–214, 1994. (page 25)

B. Lakshminarayanan, D. M. Roy, and Y. W. Teh. Top-down particle filtering for Bayesian decision trees. In *Proc. Int. Conf. Mach. Learn. (ICML)*, 2013. (pages 13 and 49)

B. Lakshminarayanan, D. M. Roy, and Y. W. Teh. Mondrian forests: Efficient online random forests. In *Adv. Neural Information Proc. Systems (NIPS)*, 2014. (pages 13 and 59)

B. Lakshminarayanan, D. M. Roy, and Y. W. Teh. Particle Gbibs for Bayesian additive decision trees. In *Int. Conf. Artificial Intelligence Stat. (AISTATS)*, 2015. (page 13)

B. Lakshminarayanan, D. M. Roy, and Y. W. Teh. Mondrian forests for large scale regression when uncertainty matters. In *Int. Conf. Artificial Intelligence Stat. (AISTATS)*, 2016. (page 13)

F. Lindsten and T. B. Schön. Backward simulation methods for Monte Carlo statistical inference. *Foundations and Trends in Machine Learning*, 6(1):1–143, 2013. (page 59)

F. Lindsten, T. Schön, and M. I. Jordan. Ancestor sampling for particle Gibbs. In *Adv. Neural Information Proc. Systems (NIPS)*, pages 2591–2599, 2012. (page 59)

F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 413–422. IEEE, 2008. (page 97)

G. Louppe and P. Geurts. Ensembles on random patches. In *Machine Learning and Knowledge Discovery in Databases*, pages 346–361. Springer, 2012. (page 98)

N. Meinshausen. Quantile regression forests. *J. Mach. Learn. Res. (JMLR)*, 7:983–999, 2006. (page 94)

T. P. Minka. Bayesian model averaging is not model combination. MIT Media Lab note. `http://research.microsoft.com/en-us/um/people/minka/papers/bma.html`, 2000. (pages 24, 40, 75, and 78)

J. Mockus, V. Tiesis, and A. Zilinskas. The application of Bayesian methods for seeking the extremum. *Towards Global Optimization*, 2(117-129):2, 1978. (page 94)

K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012. (pages 11, 81, and 85)

A. Narr, R. Triebel, and D. Cremers. Stream-based active learning for efficient and adaptive classification of 3d objects. In *Int. Conf. on Robotics and Automation*, May 2016. to appear. (page 98)

T. Nickson, M. A. Osborne, S. Reece, and S. J. Roberts. Automated machine learning on big data using stochastic algorithm tuning. *arXiv preprint arXiv:1407.7969*, 2014. (page 94)

D. Y. Pavlov, A. Gorodilov, and C. A. Brunk. Bagboo: a scalable hybrid bagging-the-boosting model. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1897–1900. ACM, 2010. (page 23)

J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., 1988. (page 85)

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and D. E. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res. (JMLR)*, 12:2825–2830, 2011. (pages 43, 75, and 89)

J. Pitman. *Combinatorial stochastic processes*, volume 32. Springer, 2006. (page 64)

M. Plummer, N. Best, K. Cowles, and K. Vines. CODA: Convergence diagnosis and output analysis for MCMC. *R news*, 6(1):7–11, 2006. (page 56)

M. Pratola. Efficient Metropolis-Hastings proposal mechanisms for Bayesian regression tree models. *arXiv preprint arXiv:1312.1895*, 2013. (pages 47 and 55)

M. T. Pratola, H. A. Chipman, J. R. Gattiker, D. M. Higdon, R. McCulloch, and W. N. Rust. Parallel Bayesian additive regression trees. *arXiv preprint arXiv:1309.1906*, 2013. (pages 47 and 52)

N. Quadrianto and Z. Ghahramani. A very simple safe-Bayesian random forest. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37:1297–1303, 2015. (page 25)

J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986. (page 26)

J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, 1993. (pages 11, 16, and 26)

J. Quinonero-Candela, C. E. Rasmussen, F. Sinz, O. Bousquet, and B. Schölkopf. Evaluating predictive uncertainty challenge. In *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, pages 1–27. Springer, 2006. (page 92)

D. M. Roy. *Computability, inference and modeling in probabilistic programming*. PhD thesis, Massachusetts Institute of Technology. (page 62)

D. M. Roy and Y. W. Teh. The Mondrian process. In *Adv. Neural Information Proc. Systems (NIPS)*, volume 21, pages 1377–1384, 2009. (pages 13, 45, 62, 67, and 82)

D. B. Rubin et al. The Bayesian bootstrap. *The annals of statistics*, 9(1):130–134, 1981. (page 25)

A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof. On-line random forests. In *Computer Vision Workshops (ICCV Workshops)*. IEEE, 2009. (pages 73, 75, and 76)

R. E. Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990. (page 12)

T. Schreiber, C. Thaele, et al. Geometry of iteration stable tessellations: Connection with Poisson hyperplanes. *Bernoulli*, 19(5A):1637–1654, 2013. (page 98)

J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Adv. Neural Information Proc. Systems (NIPS)*, 2012. (page 93)

J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Ali, and R. P. Adams. Scalable Bayesian optimization using deep neural networks. In *Proc. Int. Conf. Mach. Learn. (ICML)*, 2015. (pages 94 and 95)

D. Sorokina, R. Caruana, and M. Riedewald. Additive groves of regression trees. In *Machine Learning: ECML 2007*, pages 323–334. Springer, 2007. (pages 20 and 46)

N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proc. Int. Conf. Mach. Learn. (ICML)*, 2010. (page 93)

M. Taddy, C.-S. Chen, J. Yu, and M. Wyle. Bayesian and empirical Bayesian forests. In *Proc. Int. Conf. Mach. Learn. (ICML)*, 2015. (page 25)

M. A. Taddy, R. B. Gramacy, and N. G. Polson. Dynamic trees for learning and design. *J. Am. Stat. Assoc.*, 106(493):109–123, 2011. (pages 26, 29, 43, 74, and 78)

Y. W. Teh. A hierarchical Bayesian language model based on Pitman–Yor processes. In *Proc. 21st Int. Conf. on Comp. Ling. and 44th Ann. Meeting Assoc. Comp. Ling.*, pages 985–992. Assoc. for Comp. Ling., 2006. (pages 13, 65, and 66)

Y. W. Teh, H. Daumé III, and D. M. Roy. Bayesian agglomerative clustering with coalescents. In *Adv. Neural Information Proc. Systems (NIPS)*, volume 20, 2008. (page 29)

M. K. Titsias. Variational learning of inducing variables in sparse Gaussian processes. In *Int. Conf. Artificial Intelligence Stat. (AISTATS)*, pages 567–574, 2009. (page 89)

P. E. Utgoff. Incremental induction of decision trees. *Machine learning*, 4(2):161–186, 1989. (page 74)

D. H. Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992. (page 21)

F. Wood, C. Archambeau, J. Gasthaus, L. James, and Y. W. Teh. A stochastic memoizer for sequence data. In *Proc. Int. Conf. Mach. Learn. (ICML)*, 2009. (page 64)

X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, S. Y. Philip, et al. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, 2008. (page 11)

Y. Wu, H. Tjelmeland, and M. West. Bayesian CART: Prior specification and posterior simulation. *J. Comput. Graph. Stat.*, 16(1):44–66, 2007. (pages 26, 51, and 59)

J. L. Zhang and W. K. Härdle. The Bayesian additive classification tree applied to credit risk modelling. *Computational Statistics & Data Analysis*, 54(5):1197–1205, 2010. (page 48)