# A BART:

*(Bayesian Additive Regression Trees)*

# An Introduction

Rob McCulloch and Matt Pratola

1. Basic BART Ideas
2. A Simple Simulated Example
3. Out of Sample Prediction
4. Pros and Cons of BART
5. The BART Model and Prior
6. The BART MCMC
7. Faster Code

# 1. Basic BART Ideas

The original BART model
(Chipman, George, and McCulloch) is:

$$Y_i = f(x_i) + \epsilon_i, \quad \epsilon_i \sim N(0, \sigma^2), \ iid.$$

*where*

the function $f$ is represented as the sum of many regression trees.

BART was inspired by the Boosting literature, in particular the work of Jerry Friedman.

The connection to boosting is obvious in that the model is based on a sum of trees.

However, BART is a fundamentally different algorithm with some consequent pros and cons.

BART is a Bayesian MCMC procdure.

We:

- put a prior on the model parameters $(f, \sigma)$.
- run a Markov chain with state $(f, \sigma)$ such that stationary distribution is the posterior

$$(f, \sigma) \mid D = \{x_i, y_i\}_{i=1}^n.$$

- Examine the draws as a repesentation of the full posterior.

  In particular, we can look at marginals of $\sigma$ and $f(x)$ at any given $x$.

# 2. A Simple Simulated Example

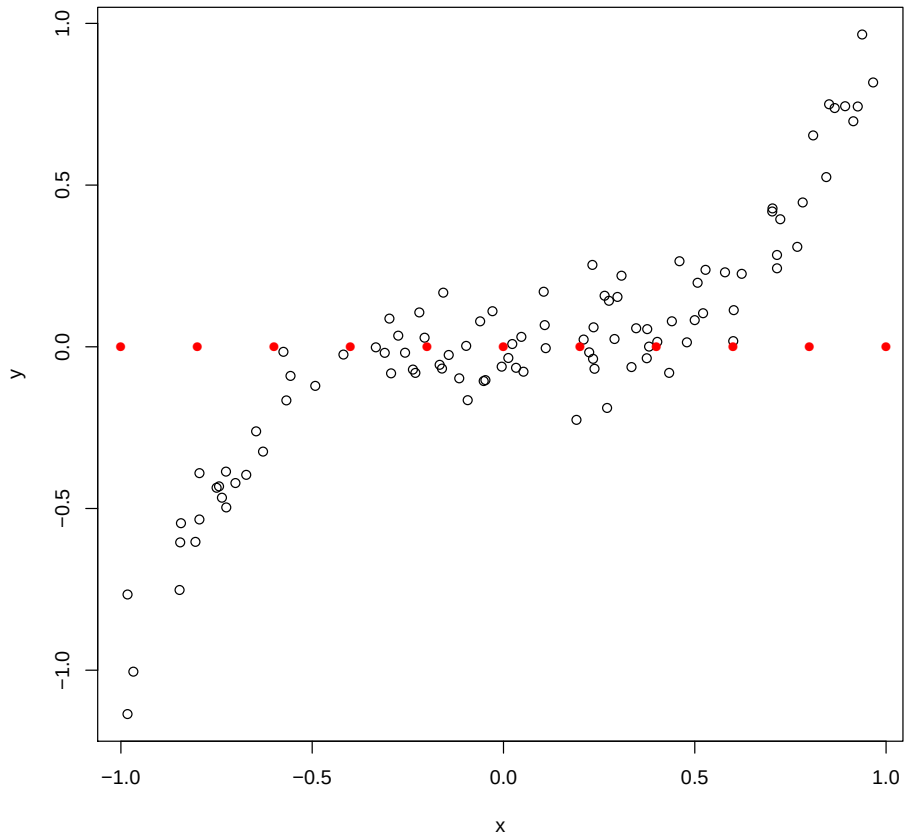
Simulate data from the model:

$$Y_i = x_i^3 + \epsilon_i \quad \epsilon_i \sim N(0, \sigma^2) \;\; iid$$

```
-----------------------------------------
n = 100
sigma = .1
f = function(x) {x^3}
set.seed(14)
x = sort(2*runif(n)-1)
y = f(x) + sigma*rnorm(n)
xtest = seq(-1,1,by=.2)
-----------------------------------------
```

Here, *xtest* will be the *out of sample* x values at which we wish to infer $f$ or make predictions.

```
-----------------------------------------------
plot(x,y)
points(xtest,rep(0,length(xtest)),col="red",pch=16)
-----------------------------------------------
```




Red is xtest.

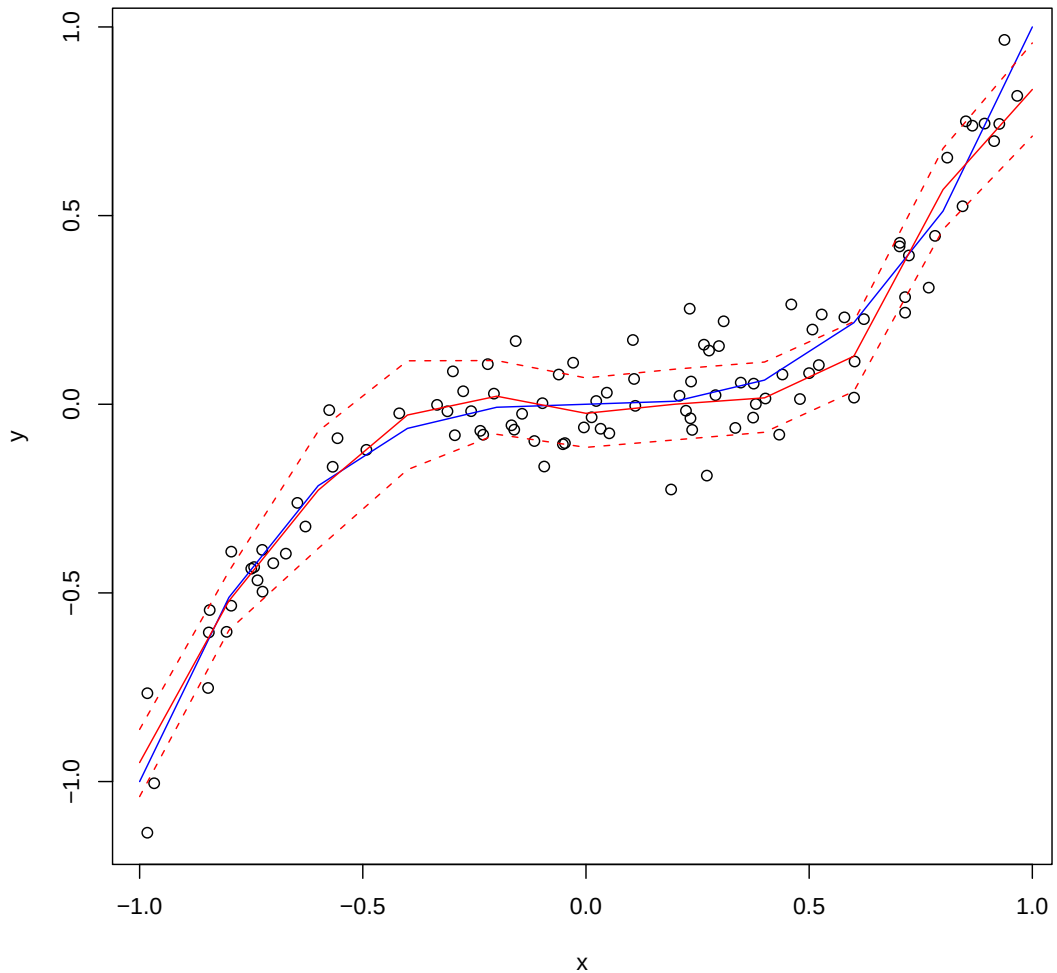
```
-------------------------------------------------
library(BayesTree)
rb = bart(x,y,xtest)
length(xtest)
[1] 11
dim(rb$yhat.test)
[1] 1000   11
-------------------------------------------------
```

The $(i, j)$ element of yhat.test is

the $i^{th}$ draw of $f$ evaluated at the $j^{th}$ value of xtest.

1,000 draws of $f$, each of which is evaluated at 11 xtest values.

```
-------------------------------------------------
plot(x,y)
lines(xtest,xtest^3,col="blue")
lines(xtest,apply(rb$yhat.test,2,mean),col="red")
qm = apply(rb$yhat.test,2,quantile,probs=c(.05,.95))
lines(xtest,qm[1,],col="red",lty=2)
lines(xtest,qm[2,],col="red",lty=2)
-------------------------------------------------
```

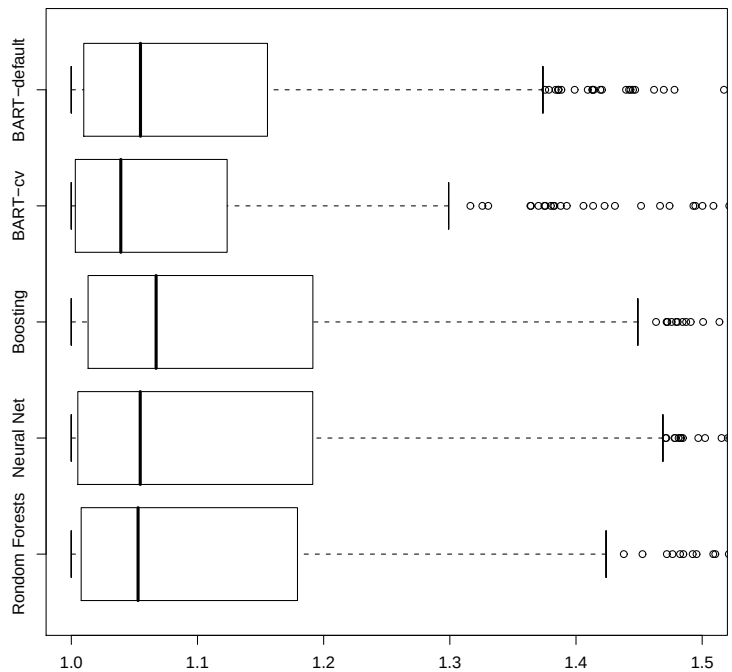


# 3. Out of Sample Prediction

Did out of sample predictive comparisons on 42 data sets.
(*thanks to Wei-Yin Loh!!*)

- p=3 − 65, $n = 100 - 7,000$.
- for each data set 20 random splits into 5/6 train and 1/6 test
- use 5-fold cross-validation on train to pick hyperparameters (except BART-default!)
- gives 20*42 = 840 **out-of-sample predictions**, for each prediction, divide rmse of different methods by the smallest

+ each boxplots represents 840 predictions for a method
+ 1.2 means you are 20% worse than the best
+ BART-cv best
+ BART-default (use default prior) does amazingly well!!



# 4. Pros and Cons of BART

## Cons:

- MCMC is slow.
- Does MCMC really converge in high dimensions?

Comments:

- New code makes BART feasible for $n$ in millions (at least).
- Ways to parallelize the computations.
- Does anything really work in high dimensions?
  BART may be competitive.

# Pros:

- As with all tree based methods, you don't have to think much about $x$.
- Size of trees (level of interaction) is not a tuning parameter. The MCMC infers the "correct" depth of the trees.
- Stochastic search.
  The flip side of "slow" is that you can get a much more extensive search of the function space than with standard boosting algorithms.
- Simple prior, given the complexity of the problem.
- Uncertainty in high dimesions.
- Can imbed many BARTs in complex models using the usual Bayesian approach.

Comments:

- Pratola's recent advance in "BART moves" may make inference and convergence much better.
- Many recent extensions: Multinomial BART, Monotonic BART, Heterskedastic BART, survival analysis, ...

# 5. The BART Model and Prior

## Regression Trees:

First, we review regression trees to set the notation for BART.

Note however that even in the simple regression tree case, our Bayesian approach is very different from the usual CART type approach.

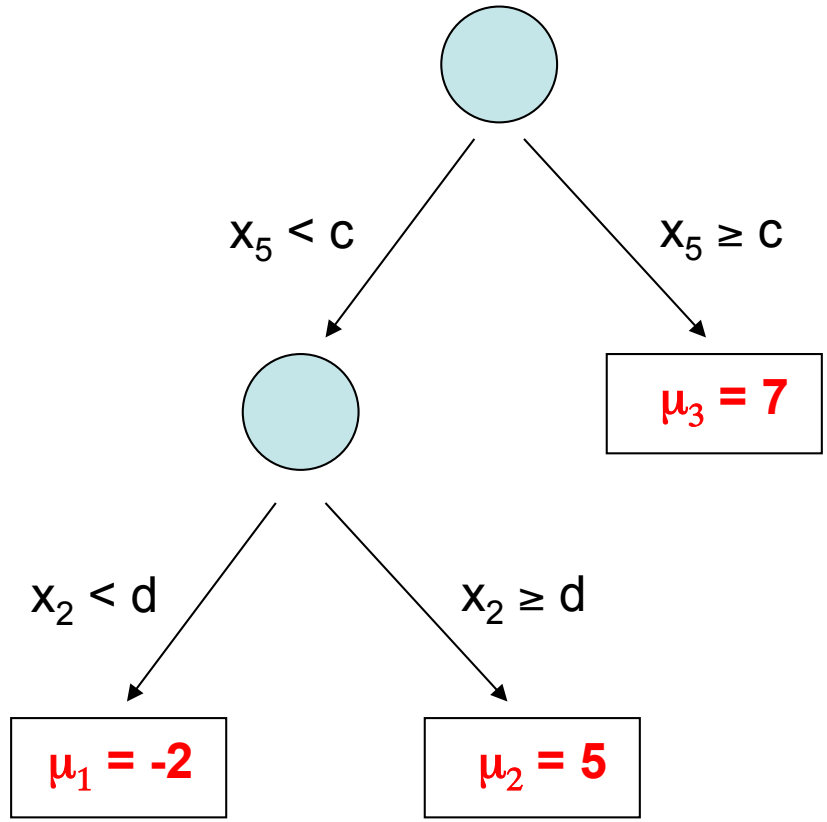
The model with have *parameters* and corresponding *priors*.

# Regression Tree:

Let $T$ denote the tree structure including the decision rules.

Let $M = \{\mu_1, \mu_2, \ldots, \mu_b\}$ denote the set of bottom node $\mu$'s.

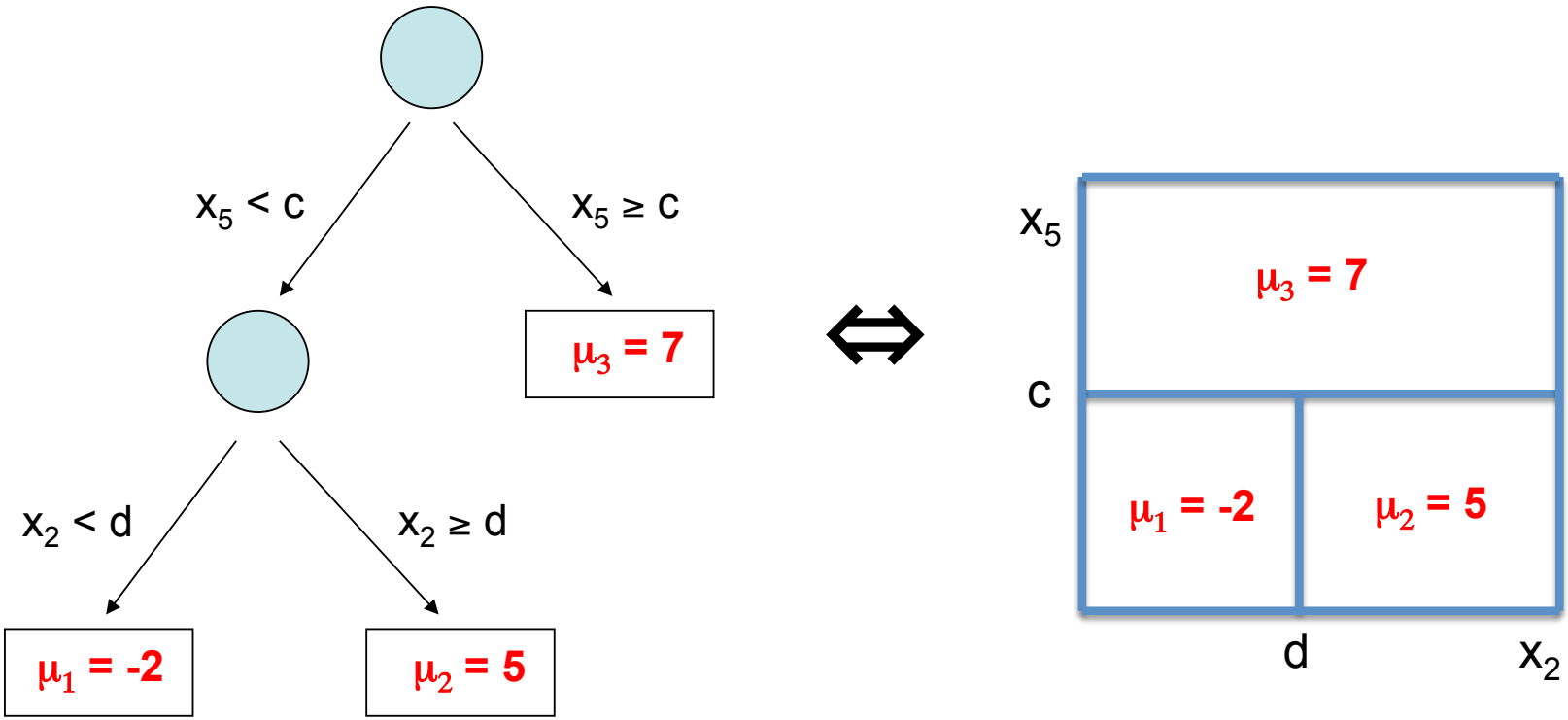
Let $g(x;\theta)$, $\theta = (T, M)$ be a regression tree function that assigns a $\mu$ value to $x$.




A single tree model:

$$y = g(x;\theta) + \epsilon.$$

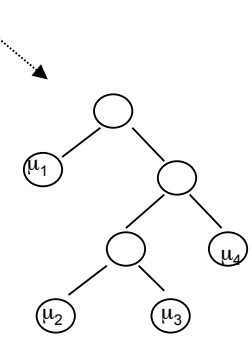
# A coordinate view of $g(x;\theta)$




Easy to see that $g(x;\theta)$ is just a step function.

# The BART Model

$$Y = g(x;T_1,M_1) + g(x;T_2,M_2) + \ldots + g(x;T_m,M_m) + \sigma z, \quad z \sim N(0,1)$$




$m = 200, 1000, \ldots,$ big, $\ldots$.

$f(x \mid \cdot)$ is the sum of all the corresponding $\mu$'s at each bottom node.

Such a model combines additive and interaction effects.

*All parameters but $\sigma$ are unidentified !!!!*

*...the connection to Boosting is obvious...*

*But,..*

Rather than simply adding in fit in an iterative scheme, we will explicitly specify a prior on the model which directly impacts the performance.

# Complete the Model with a Regularization Prior

$$\pi(\theta) = \pi((T_1, M_1), (T_2, M_2), \ldots, (T_m, M_m), \sigma).$$

$\pi$ wants:

- Each $T$ small.
- Each $\mu$ small.
- "nice" $\sigma$ (smaller than least squares estimate).

We refer to $\pi$ as a regularization prior because it restrains the overall fit.

In addition, it keeps the contribution of each $g(x; T_i, M_i)$ model component small.

# Prior on $T$

We specify a process we can use to draw a tree from the prior.

The probability a current bottom node, at depth $d$, gives birth to a left and right child is

$$\frac{\alpha}{(1+d)^{\beta}}$$

The usual BART defaults are

$$\alpha = \text{“base”} = .95, \quad \beta = \text{“power”} = 2.$$

This makes non-null but small trees likely.

```
nbottom
    1     2     3     4     5
 0.05  0.55  0.28  0.09  0.03
```

Splitting variables and cutpoints are drawn uniformly from the set of “available” ones.

# Prior on $M$

Let $\theta$ denote all the parameters.

$$f(x \mid \theta) = \mu_1 + \mu_2 + \cdots \mu_m.$$

where $\mu_i = \mu_i(x)$, is the $\mu$ in the bottom node $x$ falls to in the $i^{th}$ tree.

Let $\mu_i \sim N(0, \tau^2)$, iid.

$$f(x \mid \theta) \sim N(0, m\tau^2).$$

In practice we often, unabashadly, use the data by first centering and then choosing $\tau$ so that

$$f(x \mid \theta) \in (y_{min}, y_{max}), \text{ with high probability.}$$

This gives:

$$\tau^2 \propto \frac{1}{m}.$$

In the software, we often use the parameter $k$ to specify the $\mu$ prior via

$$\tau = \frac{max(y) - min(y)}{2\,k\,\sqrt{m}}$$

that way

$$k\,\sqrt{m}\,\tau = \frac{max(y) - min(y)}{2}$$

so that from the "middle" of $f$ to the extreme is $k$ standard deviations.

Typically, the default in the software is $k = 2$.

# Prior on $\sigma$

$$\sigma^2 \sim \frac{\nu\,\lambda}{\chi^2_\nu}$$

Default: $\nu = 3$.

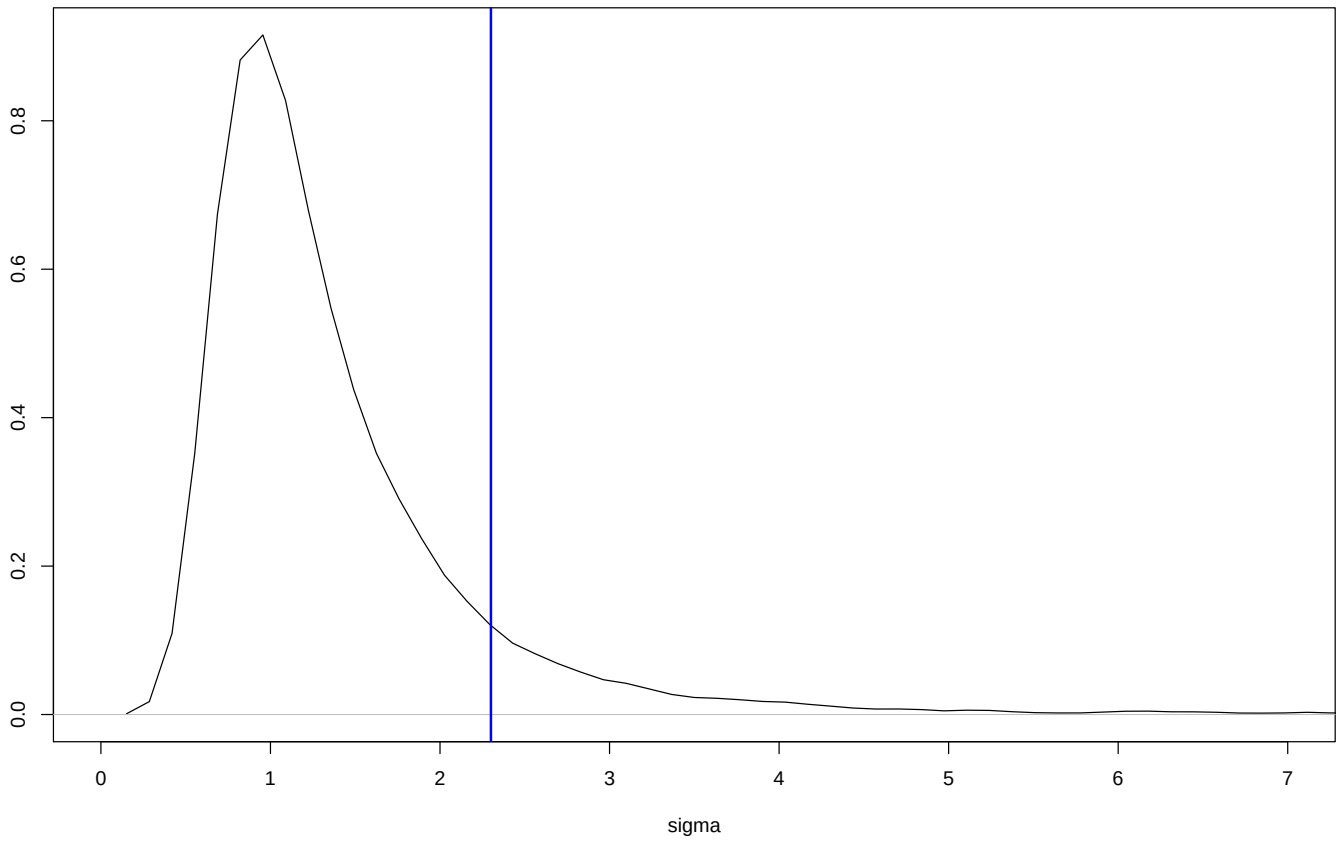
$\lambda$:

Get a reasonable estimate of $\hat{\sigma}$ of sigma then choose $\lambda$ to put $\hat{\sigma}$ at a specified quantile of the $\sigma$ prior.

Default: quantile $= .9$

Default: if $p < n$, $\hat{\sigma}$ is the usual least squares estimate, else $sd(y)$.

Solid blue line at $\hat{\sigma}$.




Conjecture: Most "failures" of BART are due to this default.

From the old BayesTree package, arguments for bart():

```
function (x.train, y.train, x.test = matrix(0, 0, 0), sigest = NA,
    sigdf = 3, sigquant = 0.9, k = 2, power = 2, base = 0.95,
    binaryOffset = 0, ntree = 200, ndpost = 1000, nskip = 100,
    printevery = 100, keepevery = 1, keeptrainfits = TRUE, usequants = FALSE,
    numcut = 100, printcutoffs = 0, verbose = TRUE)
```

# 6. The BART MCMC

$$Y = g(x;T_1,M_1) + \ldots + g(x;T_m,M_m) + \sigma z$$
plus
$$\pi((T_1,M_1),\ldots.(T_m,M_m),\sigma)$$

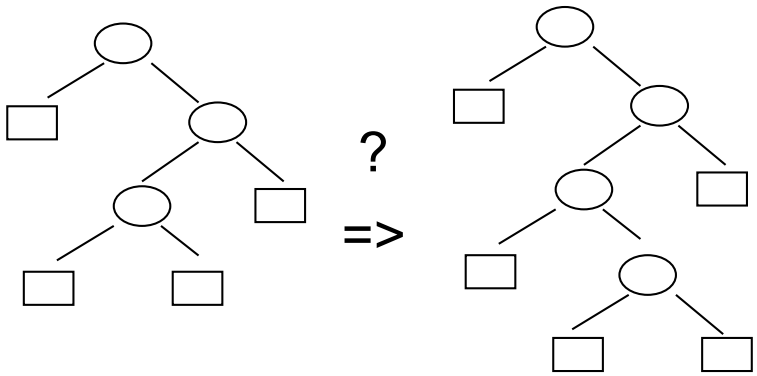
First, it is a "simple" Gibbs sampler:

$$\begin{aligned} (T_i, M_i) \quad &| \quad (T_1, M_1, \ldots, T_{i-1}, M_{i-1}, T_{i+1}, M_{i+1}, \ldots, T_m, M_m, \sigma) \\ \sigma \quad &| \quad (T_1, M_1, \ldots, \ldots, T_m, M_m) \end{aligned}$$

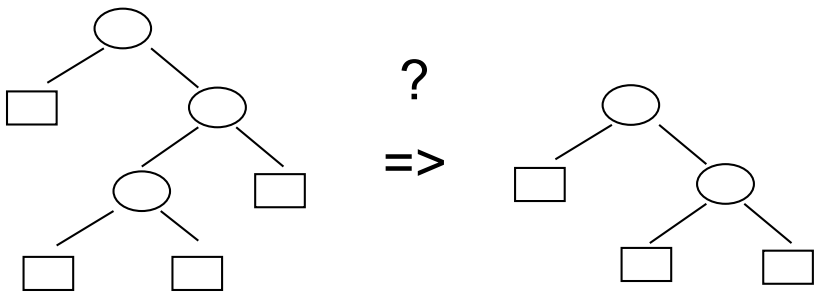
To draw $(T_i, M_i) \mid \cdot$ we subract the contributions of the other trees from both sides to get a simple one-tree model.

We integrate out $M$ to draw $T$ and then draw $M \mid T$.

To draw $T$ we use a Metropolis-Hastings with Gibbs step.
We use various moves, but the key is a "birth-death" step.




propose a more complex tree




propose a simpler tree

$$Y = g(x;T_1,M_1) + \ldots + g(x;T_m,M_m) + \sigma z$$
plus
$$\pi((T_1,M_1),\ldots.(T_m,M_m),\sigma)$$

## Connections to Other Modeling Ideas:

Bayesian Nonparametrics:

- Lots of parameters to make model flexible.
- A strong prior to shrink towards a simple structure.
- BART shrinks towards additive models with some interaction.

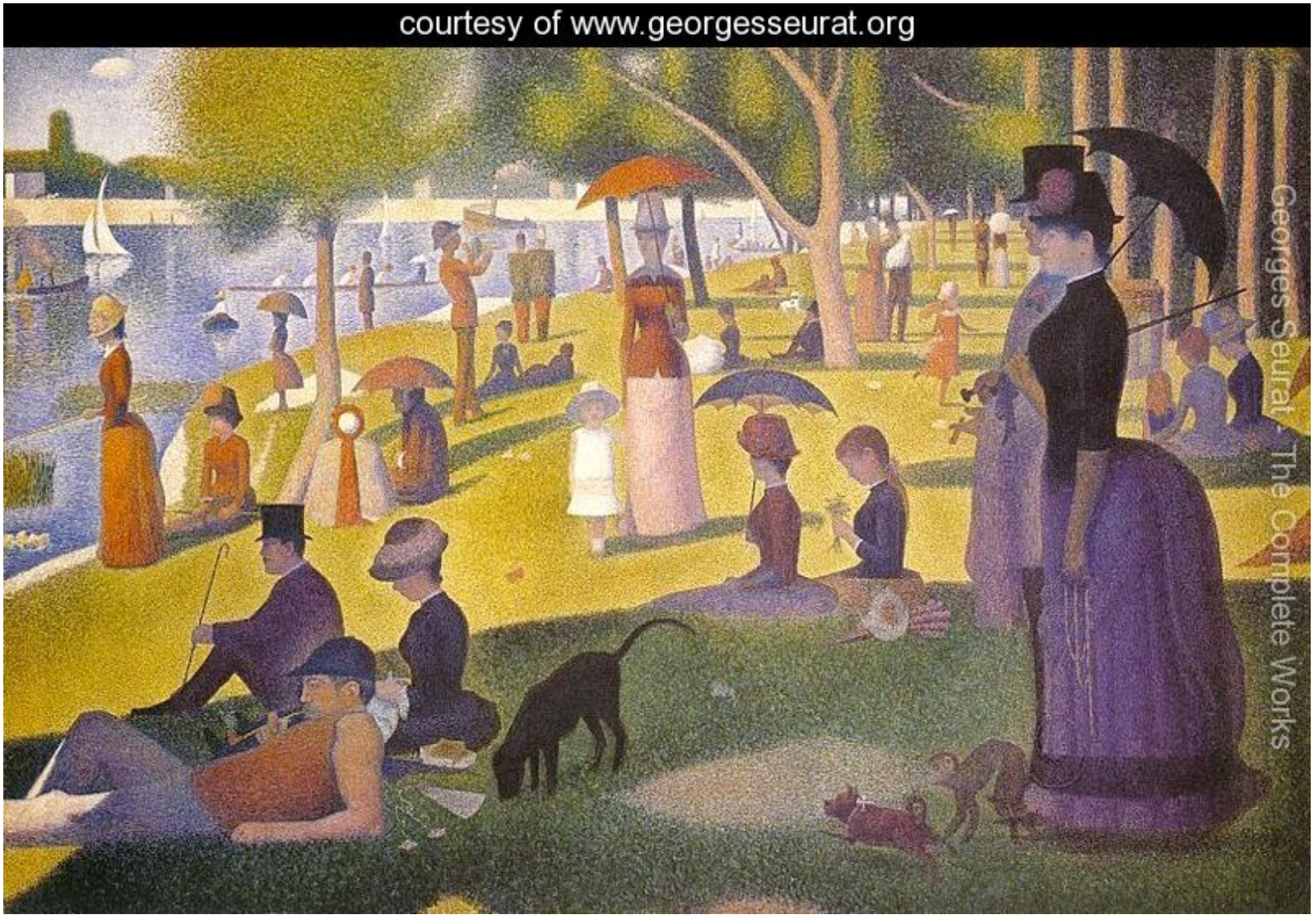
Dynamic Random Basis:

- $g(x; T_1, M_1), g(x; T_2, M_2), \ldots, g(x; T_m, M_m)$ are dimensionally adaptive.

Gradient Boosting:

- Overall fit becomes the cumulative effort of many weak learners.

*Build up the fit, by adding up tiny bits of fit ..*



# 7. Faster Code

1. Rewrote code so that it is leaner.

2. Used MPI to compute.

| | num obs | new-parallel | new-serial | old |
|---|---|---|---|---|
| 1 | 1000 | 7 | 9 | 43 |
| 2 | 2000 | 8 | 18 | 95 |
| 3 | 3000 | 9 | 28 | 149 |
| 4 | 4000 | 10 | 36 | 204 |
| 5 | 5000 | 12 | 45 | 262 |
| 6 | 10000 | 18 | 90 | 547 |
| 7 | 50000 | 70 | 439 | NA |
| 8 | 100000 | 138 | 902 | NA |
| 9 | 500000 | 904 | 6410 | NA |

With 10,000 observations the new algorithm is $547/90 = 6$ times faster than the old algorithm.

The parallel version is $90/18 = 5$ times faster than the serial version (with 7 cores).

Thus, the parallelized new algorithm is 30 times faster than the old BART algorithm (available in the R package BayesTree).

With 500,000 observations, the old algorithm cannot be run on the machine being used. The parallel version is $6410/904 = 7$ times faster than the serial version. Recall that we are using 7 cores to do the basic computations.