Introduction to computing with BART

Rodney Sparapani
Medical College of Wisconsin

September 30: BART Bootcamp
Biostatistics in the Modern Computing Era
Medical College of Wisconsin, Milwaukee campus

Outline

- Installing the **BART** R package
- Comparison of BART R packages on CRAN
- BART, ensembles and prediction error
- A brief history and overview of multi-threaded computing
- Multi-threading with the **BART** R package
- Live demonstration of multi-threading with **BART**
- Creating a BART executable with C++ sans R

Installing our BART R package

- Our **BART** R package (current version 1.3) is on the Comprehensive R Archive Network (CRAN)
- `https://cran.r-project.org/package=BART`
- Install into your personal vs. global R library
- `~/.Rprofile`
- `# my .Rprofile contains this personal library`
- `.libPaths("~/R/3.4.0/lib64/R/library")`
- Installing **BART** (which depends on the Rcpp package)
- From source with the Unix command line (from here on Unix means UNIX/Linux/macOS)
- Requires a full C++ toolchain like GNU GCC or Apple Xcode
- `$ R CMD INSTALL BART_1.3.tar.gz`
- From the R prompt for Windows and Unix
- `> options(repos=c(`
- `+ CRAN="https://cran.r-project.org"))`
- `> install.packages("BART", dependencies=TRUE)`

BART R packages on CRAN comparison

	dbarts	**BART**	**bartMachine**
Author(s)	Dorie	McCulloch	Kapelner
		Sparapani	Bleich
Computer language	C++	C++	java
Dependencies	None	Rcpp	rJava
Multi-threaded	No	Yes	Yes
`predict` function	No	Yes	Yes
Missing data handling	No	No*	Yes
Variable selection	No	No*	Yes
Tree transition proposals	4	3	3
Partial dependence plots	Yes	No*	Yes
Continuous & binary	Yes	Yes	Yes
Time-to-event	No	Yes	No
Convergence diagnostics	Continuous	All	Continuous
Thinning	Yes	Yes	No
Cross-validation	Yes	No*	Yes

*you have learned, or will learn, how to do these today

BART, ensembles and prediction error

- mean squared error $=$ bias^2 $+$ variance
- There is a trade-off between the bias and variance
- Consider the spectrum of trade-offs
- Linear regression is on the high bias/low variance end
- Single-tree regression is on the low bias/high variance end
- Ensembles are in the middle: medium bias/medium variance
- BART is in the class of ensemble models which both theoretically, and in practice, have excellent out-of-sample predictive performance

Krogh & Solich 1997 *Physical Review E*
Baldi & Brunak 2001 “Bioinformatics: machine learning approach”
Kuhn & Johnson 2013 “Applied Predictive Modeling”

A brief history of multi-threading

- 1961: Burroughs B5000 Asymmetric Multiprocessing
- 1962: Burroughs D825 Symmetric Multiprocessing (SMP)
- 1967: Amdahl's law Gain=$\left((1-b)/C+b\right)^{-1}$
- 2000: AMD64 architecure debuts: native execution of 32-bit x86 legacy code as well as new 64-bit x86 instructions
- 2003: Linux kernel 2.6 unleashes SMP support
- 2005: AMD Opteron dual core chips debut
- 2007: AMD Opteron 4 core chips debut
- 2008: Intel Xeon 4 cores (8 threads) debut
- 2009: AMD Opteron 6 core chips debut
- 2010: AMD Opteron 12 core chips debut
- 2010: Intel Xeon 8 cores (16 threads) debut
- 2011: AMD Opteron 16 core chips debut

Modern multi-threading software frameworks

- Message Passing Interface (MPI) for multiple nodes
- Pratola, Chipman, Gattiker, Higdon, McCulloch, Rust. Parallel Bayesian Additive Regression Trees. JCGS 2014;23:830-852 `https://arxiv.org/abs/1309.1906`
- OpenMP for single nodes: used by **BART** for `predict`
- detected by the GNU autotools when **BART** installed
- defines a C pre-processor macro (or not): `_OPENMP`
- not for Windows: GNU autotools not available (Cygwin)
- not for macOS: not supported by Apple Xcode (clang)
- Forking for single nodes: **parallel** R package
- Forking not supported on Windows
- see the help page: `?mcparallel`
- Forking used by **BART** for posterior sampling
- `mc.wbart`, `mc.pbart`, `mc.surv.bart`, `mc.recur.bart`
- can be used by the `predict` function instead of OpenMP

Multi-threading: can I run multiple threads?

- Windows: not currently supported by R or CRAN
- Unix only at this point
- OpenMP

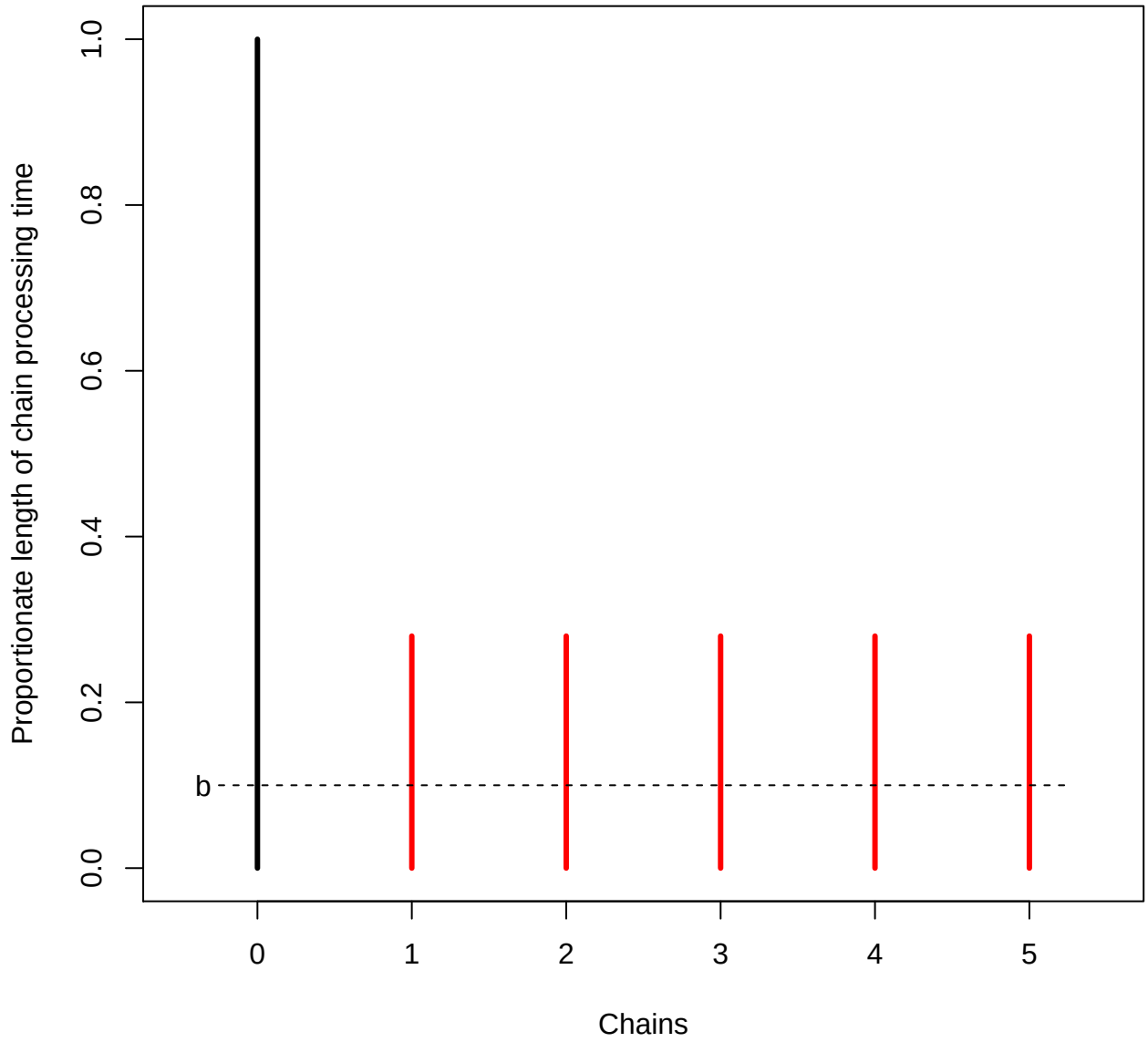
```
> library(BART)
> mc.cores.openmp()
```

- Returns 0 if OpenMP not available; 1 if it is
- Forking

```
> library(parallel)
> detectCores()
```

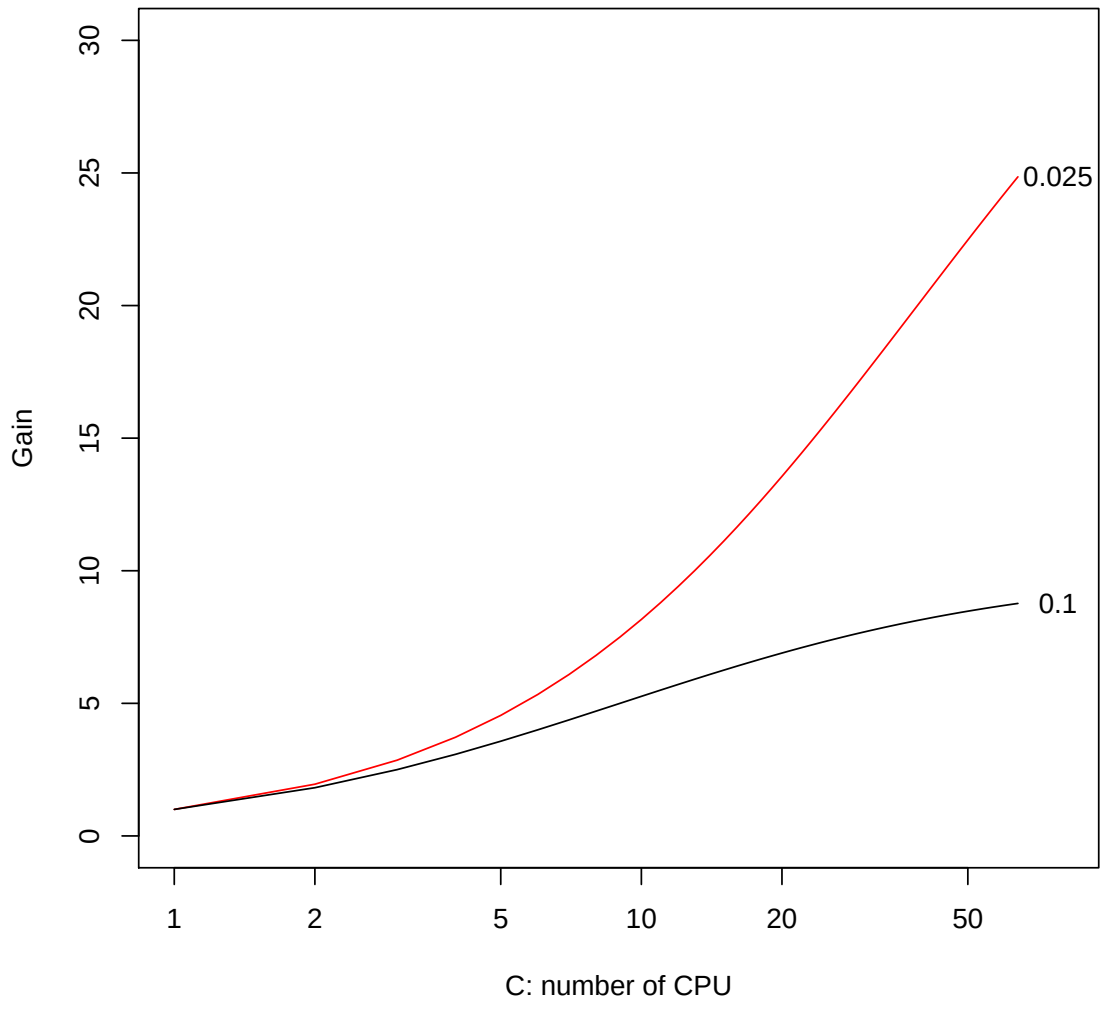
- Returns 1 or more (and occasionally NA)

MCMC is “embarrassingly” parallel



Amdahl's Law and the MCMC Corollary

- Gain= $\frac{1-b+b}{(1-b)/C+b} = \frac{1}{(1-b)/C+b}$ and b is the burn-in fraction



Multi-threading: random access memory (RAM) I

- IEEE 754 specifies that every double-precision number consumes 8 bytes (64 bits) so you can estimate your needs
- If A is $m \times n$, then $\mathbf{RAM}(A) = 8 \times m \times n$ bytes
- If you consume all of the physical RAM, the system will "swap" segments out to virtual RAM which are disk files:
  this will degrade performance and possibly crash the system
- On Unix, you can monitor memory and swap usage with `top`
- Within R, you can determine the size of an object with the `object.size` function

Multi-threading: random access memory (RAM) II

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

- R matrices are column-major: $[a_{11}, a_{21}, \ldots, a_{12}, a_{22}, \ldots]$
- C++ matrices are row-major: $[a_{11}, a_{12}, \ldots, a_{21}, a_{22}, \ldots]$
- This is easily addressed with a transpose
  instead of passing A from R to C++, we pass A^t
- R passes objects by pointer, but it is copy-on-write
- All objects in the parent thread can be read by the child thread from the pointer without a copy, but when an object is altered/written by the child, then a new copy is created
- $\mathbf{RAM}(A) = 8 \times m \times n \times C$ and C is the no. of children
- If the parent transposes, we avoid the copy: `A <- t(A)`

Multi-threading: interactive vs. batch processing

- Interactive jobs must take precedence over batch jobs to prevent the user experience from suffering high latency
- Examples of interactive activity: typing at the command line, editing files with Emacs, reading email, browsing the web
- In the **tools** R package, there is the `psnice` function
- Paraphrased from the `?psnice` help page

  > *Unix has a concept of process priority. Priority is assigned values from 0 to 39 with 20 being the normal priority and (counter-intuitively) larger numeric values denoting lower priority. Adding to the complexity, there is a "nice" value, the amount by which the priority exceeds 20. Processes with higher nice values will receive less CPU time than those with normal priority. Generally, processes with nice 19 are only run when the system would otherwise be idle.*

- by default, the **BART** package children have nice set to 19

`wbart` and `mc.wbart` input and output

```
post <- wbart(x.train, y.train, ...,
        ndpost=M, keepevery=1) or
post <- mc.wbart(x.train, y.train, ...,
        ndpost=M, keepevery=1, mc.cores=2, seed=99)
```

Input matrices: `x.train` and, optionally, `x.test`: x_i

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

Output object, `post`, of type `wbart` which is essentially a list

Matrices: `post$yhat.train` and `post$yhat.test`: $\hat{y}_{im} = f_m(x_i)$

$$\begin{bmatrix} \hat{y}_{11} & \cdots & \hat{y}_{N1} \\ \vdots & \vdots & \vdots \\ \hat{y}_{1M} & \cdots & \hat{y}_{NM} \end{bmatrix}$$

`predict` input and output

`pred <- predict(post, x.test, mc.cores=1, ...)`

`post` object of type `wbart` (continuous), `pbart` (binary probit), `lbart` (binary logistic), `survbart` (survival analysis), `criskbart` (competing risks) or `recurbart` (recurrent events)

Input matrices: `x.test`: x_i

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_Q \end{bmatrix}$$

Output matrix for `wbart`, `pbart` and `lbart`: $\hat{y}_{im} = f_m(x_i)$

$$\begin{bmatrix} \hat{y}_{11} & \cdots & \hat{y}_{Q1} \\ \vdots & \vdots & \vdots \\ \hat{y}_{1M} & \cdots & \hat{y}_{QM} \end{bmatrix}$$

The parallel R package and `mc.wbart`

```
mc.wbart <- function(..., nice=19, transposed=FALSE) {
RNGkind("L'Ecuyer-CMRG")
set.seed(seed)
parallel::mc.reset.stream()
if(!transposed) {
    x.train <- t(x.train)
    x.test <- t(x.test)
}
mc.cores <- min(c(mc.cores, parallel::detectCores()))
...
for(i in 1:mc.cores)
        parallel::mcparallel({psnice(value=nice);
                    wbart(..., transposed=TRUE)},
                    silent=(i!=1))
                    ## to avoid duplication of output
                    ## capture from first child only
post.list <- parallel::mccollect()
```

Multi-threading live demonstration

- With the `system.file` function, you can find where R installed the **BART** package as well as files and sub-directories
- `system.file(package='BART')`
- `system.file('demo', package='BART')`
- `system.file('demo/friedman.wbart.R', package='BART')`
- For the demos, you can use the `demo` function
- `demo(package='BART')`
- `demo('friedman.wbart', package='BART')`
- But then you have to press Return after each plot

Creating a BART executable with C++ sans R

- The **Rcpp** package is a dependency for the **BART** package
- **Rcpp** provides a seamless transition by passing object pointers from R to C++ and back again
- **Rcpp** allows C++ code to rely on the R RNG
- In our package, you can build C++ BART without R/**Rcpp**
- C++ BART is built with the standalone Rmath library which is part of the R project and contained in the R source code
- Rmath provides an R compliant RNG and all of the useful R functions like `pnorm`
- You can optionally build with the RNG provided by the C++ `random` class from the Standard Template Library (STL)
- `system.file('cxx-ex', package='BART')`
- `system.file('cxx-ex/Makefile', package='BART')`