# The Arborist: a Scalable Decision Tree Implementation

Mark Seligman

Rapidics LLC

July 1, 2014

- Outline
- 2 Introduction
- Random Forests
- Arborist
- 6 Implementation
- 6 Summary and future work

#### Outline

- Introduction
- Random Forests
- Arborist
- Implementation
- Summary and future work
- Q & A

- Outline
- 2 Introduction
- Random Forests
- Arborist
- 6 Implementation
- 6 Summary and future work

- Rapidics: performance as a service.
- Began with customer inquiry: can graphics processors accelerate decision tree algorithms?
- Examination of existing implementations, literature revealed low-hanging fruit for performance improvements.
- Implemented new, whole-cloth version employing these ideas, as well as our own. Initial goal was entry from  $\mathbf{R}$ .
- Attention to modularity leaves options open for extending functionality: multiple front ends, anticipating workflows, usw.
- Opportunities for multicore acceleration reveal themselves as algorithmic structure flushed out.
- Coprocessor (GPU) opportunities become apparent as work progresses.

- Outline
- 2 Introduction
- Random Forests
- Arborist
- Implementation
- 6 Summary and future work

6 / 31

# Decision trees, briefly

- Prediction method involving, intuitively, a series of yes/no questions.
- Answer to each question determines which (of two) questions to pose next.
- Each series of questions can be represented as a binary tree, whence the name.
- A training phase builds the trees from observational data, assigning prediction scores to the leaves.
- A prediction phase walks the trees on test data, deriving an overall prediction from the score reported by the terminal leaf of each tree.

## Random Forest algorithm

- Trademarked name registered to Leo Breiman (dec.) and Adele Cutler.
- Binary decision tree method for either regression or classification.
  - ▶ Regression predicts the mean of numerical-valued response.
  - Classification predicts the category having a plurality.
- Observational data ("predictors") can include both numerical and categorical values.
- Trains on randomly-selected subset of the observations ("bagging").
- Prediction performed on held-out subset as well as separately-provided test sets.
- Individual trees built from a series of splitting decisions.

8 / 31

### RF algorithm, cont.

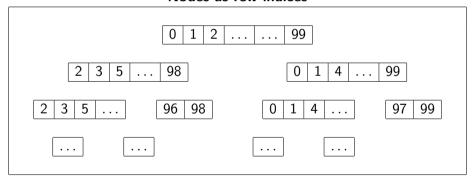
- It is helpful to think of tree nodes as sets of row indices, with the root node consisting of all sampled rows.
- Branching ("splitting") criterion for a node consists of a predictor and either an order relation (continuous) or a membership relation (factor) on the predictor.
- For each node, candidate splitting predictors are a random subsample of the full set.
- The criterion expresses which predictor and which subset or cutpoint within observerations on that predictor, over the node, maximize a measure of informational gradient, typically Gini gain.
- The splitting criterion therefore defines a bipartition of observations. The rows
  corresponding to these observations determine the left and right descendants of the
  current node in the decision tree.
- Splitting halts when size or informational constraint reached.



### Example: 100 rows split as unbalanced tree: four levels, six leaves

• The leaves at the penultimate level may be the result of either an inability to maximize Gini gain or a constraint on minimal node size.

#### Nodes as row indices

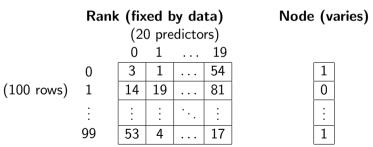


## Node splitting

- For a given node, each selected predictor is walked in its respective order.
- Information metric computed at each step, retaining position of global maximum.
- Note the computational expense, viz., memory traffic: rows visited within the node are the same for all predictors, but the order varies with the individual predictors.
- Note that although predictor orderings are fixed for a given data set, node membership of a given row varies across trees as well as down the levels of a single tree.
- The implementational trick is to retain this ordering economically. Expensive solution would be to resort on each subtree.

# Walking the rows in a node, by predictor

- The predictor ranks are fixed by the obervational data, but row-to-node assignments change from level to level.
- The rows are visited in distinctly different orders, depending on both the predictor and the node.



- Outline
- 2 Introduction
- Random Forests
- 4 Arborist
- 6 Implementation
- 6 Summary and future work

#### The Arborist

- Commercial implementation of RF, geared to commodity high-performance hardware as well as customizable configuration within specialized applications.
- Deployed implementations exploit large-memory ("medium data"), multicore and multiprocessor platforms.
- Sample, row and predictor weighting, as well as quantile regression, all available as standard features.
- "Loopless" reinvocation feature to enable iterative calls while bypassing reinitialization.
- Bindings available for **R** language using **Rcpp** C++ template extensions, although the implmentation is essentially front-end agnostic.
- **Python**, **Julia** bindings anticipated, to encourage adoption by larger ML community and to facilitate comparison with other accelerated RF implementations.

## Aside: performance is data-dependent

- As with linear algebra, the appropriate treatment depends on the contents of the (design) matrix: SVD, for example.
- E.g., is response continuous or categorical? What about the predictors?
- Constraints in response or predictor values may benefit from numerical simplification.
- Are we choosing sparse predictor subsets? typical of categorical response.
- Will ties play a significant role? if not, can avoid tracking them.
- Custom implementations rely heavily on the answer to such questions.

- Outline
- 2 Introduction
- Random Forests
- Arborist
- 6 Implementation
- 6 Summary and future work

# Low-hanging fruit

- Compiled code (here, **Rcpp**), to stage data.
- Since splitting is based on ordering or subset membership, observations are pre-sorted.
- Calls to random-variate generators hoisted out of inner loops or replaced by jittering.
- Retention of leaf maps, as opposed merely to scores, from each tree permits quantile evaluation.
- Permutation of response vector permits "loopless" (i.e., from the user's perspective) resampling.

## Opportunities for parallelization

- Within a node, predictors can be evaluated in parallel.
- Nodes can themselves be evaluated in parallel, although we have found it more profitable to evaluate entire levels sequentially.
- Trees can be built and walked independently a common Hadoop-based strategy, for example.
- A key synchronization point, however, lies in passing from node to descendants: splitting.
   Split determination remains the slow step.
- At first blush, memory accesses within a node appears to be purely random. In fact, can
  be implemented as a series of scatters which become progressively more local further
  down the tree. This observation turns out to be essential in efforts to improve
  performance of splitting.

## Splitting via state accumulators

- Walks entire predictor column in order, noting corresponding row index.
- Looks up node associated with row and loads its accumulator.
- Updates information metric and writes accumulator.
- Drawback: undergoes considerable memory traffic at each update.
- State access is irregular, does not benefit from improved data locality as tree levels descend.

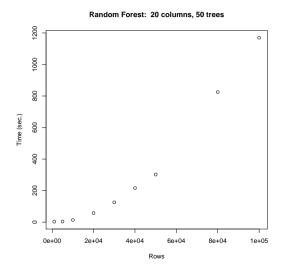
## Restaging + splitting

- "Restaging": maintains sorted response vectors, one per node per predictor, followed by separate splitting pass.
- As levels descend, elements of sorted vector sent either to left or right according to a
  predicate: algorithm known in engineering circles as a 'stable partition'. Order preserved
  in the two subnodes, so resorting unnecessary.
- Number of vectors grows with node count: up to  $2^{I}$  at level I, but individual lengths diminish.
- Splitting proceeds in a separate pass from restaging, but now memory access is regular: unit-stride vectors of small slot size.
- Data locality improves with level because displacement to L or R is shrinking.
- This approach chosen for recent incarnations of Arborist, especially due to salutary benefits for GPU.

#### Performance

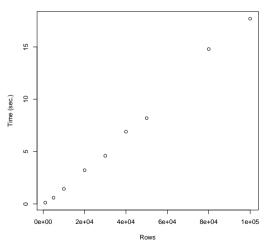
- Transfer from front end, plus presort: may preponderate at small tree count, but tends to amortize with a few hundred trees.
- Tree-walking (prediction) scales well with core count (OpenMP). Also on the order or 1%.
   GPU schemes have been devised and reported elsewhere.
- Gini computations scale well with core count.
- Restaging does not scale with core count, likely owing to dearth of data locality closer to the root: frequent TLB misses seen at high row count.
- Restaging is typically the biggest performance hurdle.

# Friedman-like benchmark adapated from MLBench

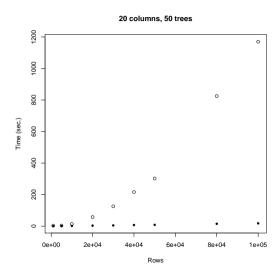


#### Same benchmark

Aborist: 20 columns, 50 trees



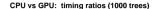
# Overlay

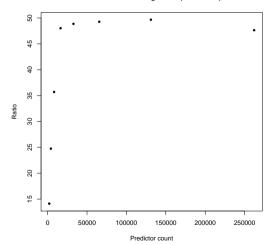


# GPU: pilot study with University of Washington team

- $\bullet$  GWAS data provided by UW: 100 samples, up to  $\sim 10^6$  predictors. Binary response, 3-level factor-valued predictors.
- Specialized version of Arborist spun off in about one day, RbSNP, exploiting fixed properties of the data: roughly 3x over Arborist.
- Apply successive perturbations to RbSNP and examine effect on performance.
- Begin by exploiting predictor-level parallelism, as on CPU.
- Ultimately restaging, Gini, argmax, next-level setup all performed on GPU. Controlling loop on CPU.

#### **RbSNP** vs. GPU: ratios





#### GPU: observations

- Major bottleneck exposed handling data.frame in compiled code.
- Throughput may be sufficient to permit pipelining of training, prediction.
- Don't need 10<sup>6</sup> predictors? parallelize across trees.
- Dynamic parallelism may provide additional benefits in the case of factor-valued predictors: load balancing.
- Due to shared-memory constraints, scaling to high sample count may entail hybrid schemes.

- Outline
- 2 Introduction
- Random Forests
- Arboris
- 6 Implementation
- 6 Summary and future work

# **Availability**

- Recent version can be run on PaaS vendor Nimbix.
- CRAN package to be dubbed Rborist.
- Source base on GitHub: suiji/Arborist.
- **Arborist** directory contains core C++ library.
- BridgeR contains Rccp and RccpArmadillo front-end bridge and call-backs.
- Call-backs mainly implement sampling and statistical functionality.
- BridgeR licensed under GPL, Arborist under MPL-2.

## Summary

- Numerous opportunities exist to accelerate decision tree construction.
- Which opportunities are best exploited depend critically on the characteristics of the data under consideration: dispatch.
- Initialization, especially when **data.frame** involved, poses a significant cost for larger data sets, but this tends to be amortized by tree count.
- Prediction, while treatable by GPUs, is not a major contributor to run time.
- Floating-point arithmetic and split determination is also not a major contributor.
- Node partitioning is the major bottleneck for execution time.
- Significant speedup can be realized by recasting the splitting pass into two separate stages.

#### Future work

- Complete source roll-out onto GitHub site, CRAN.
- Address TLB miss rate for CPU version by means of "huge" pages.
- Blocking techniques to accommodate higher sample sizes and out-of-coprocessor memory loads: graduate from medium to big data.
- Integrate with multi-node schemes to exploit cloud, data-center resources.
- Further anticipation of workflows.