Big Data Decision Trees with R

Author Richard Calaway, Lee Edlefsen, and Lixin Gong
The included rxDTree function provides the ability to estimate decision trees efficiently on very large data sets. Decision trees (*Breiman, Friedman, Olshen, & Stone, 1984*) provide relatively easy-to-interpret models, and are widely used in a variety of disciplines. For example,

Predicting which patient characteristics are associated with high risk of, for example, heart attack. Deciding whether or not to offer a loan to an individual based on individual characteristics. Predicting the rate of return of various investment strategies

The rxDTree function fits tree models using a binning-based recursive partitioning algorithm. The resulting model is similar to that produced by the recommended R package rpart (*Therneau & Atkinson, 1997*). Both classification-type trees and regression-type trees are supported.The rxDTree AlgorithmDecision trees are effective algorithms widely used for classification and regression. Classical algorithms for building a decision tree sort all continuous variables in order to decide where to split the data. This sorting step becomes time and memory prohibitive when dealing with large data. Various techniques have been proposed to overcome the sorting obstacle, which can be roughly classified into two groups: performing data pre-sorting or using approximate summary statistics of the data. While pre-sorting techniques follow classical decision tree algorithms more closely, they cannot accommodate very large data sets.

These big data decision trees are normally parallelized in various ways to enable large scale learning: data parallelism partitions the data either horizontally or vertically so that different processors see different observations or variables and task parallelism builds different tree nodes on different processors.The rxDTree algorithm is an approximate decision tree algorithm with horizontal data parallelism, especially designed for handling very large data sets. It computes histograms to create empirical distribution functions of the data and builds the decision tree in a breadth-first fashion.

The algorithm can be executed in parallel settings such as a multicore machine or a distributed (cluster or grid) environment. Each worker gets only a subset of the observations of the data, but has a view of the complete tree built so far. It builds a histogram from the observations it sees, which essentially compresses the data to a fixed amount of memory. This approximate description of the data is then sent to a master with constant low communication complexity independent of the size of the data set. The master integrates the information received from each of the workers and determines which terminal tree nodes to split and how. Since the histogram is built in parallel, it can be quickly constructed even for extremely large data sets.With rxDTree, you can control the balance between time complexity and prediction accuracy by specifying the maximum number of bins for the histogram.

The algorithm builds the histogram with roughly equal number of observations in each bin and takes the boundaries of the bins as the candidate splits for the terminal tree nodes. Since only a limited number of split locations are examined, it is possible that a suboptimal split point is chosen causing the entire tree to be different from the one constructed by a classical algorithm. However, it has been shown analytically that the error rate of the parallel tree approaches the error rate of the serial tree, even though the trees are not identical (*Ben-Haim & Tom-Tov, 2010*). You can set the number of bins in the histograms to control the tradeoff between accuracy and speed: a large number of bins allows a more accurate description of the data and thus more accurate results, whereas a small number of bins reduces time complexity and memory usage.In the case of integer predictors for which the number of bins equals or exceeds the number of unique observations, the rxDTree algorithm produces the same results as classical sorting algorithms because the empirical distribution function exactly represents the data set...