REVOLUTION ANALYTICS WHITE PAPER

Big Data Decision Trees with R

By Richard Calaway, Lee Edlefsen, and Lixin Gong

Fast, Scalable, Distributable Decision Trees

Revolution Analytics' RevoScaleR package provides full-featured, fast, scalable, distributable predictive data analytics. 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 Algorithm

Decision 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

A Simple Classification Tree

An example from the `rpart` package can be easily adapted to the `rxDTree` function. The kyphosis data set contains data on children who have had corrective spinal surgery. The dependent variable indicates whether or not a type of spinal deformation (Kyphosis) was present after the surgery. Independent variables are Age (in months), `Start` (the number of the first vertebra operated on), and `Number` (the number of vertebrae involved). The classification tree can be estimated with `rxDTree` as follows:

```
data("kyphosis", package="rpart")
kyphTree <- rxDTree(Kyphosis ~ Age + Start + Number,
        data = kyphosis,  cp=0.01)
kyphTree
Call:
rxDTree(formula = Kyphosis ~ Age + Start + Number, data = kyphosis,
    cp = 0.01)
Data: kyphosis
Number of valid observations:  81
Number of missing observations:  0

Tree representation:
n= 81

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 81 17 absent (0.79012346 0.20987654)
   2) Start>=8.5 62  6 absent (0.90322581 0.09677419)
     4) Start>=14.5 29  0 absent (1.00000000 0.00000000) *
     5) Start< 14.5 33  6 absent (0.81818182 0.18181818)
      10) Age< 55 12  0 absent (1.00000000 0.00000000) *
      11) Age>=55 21  6 absent (0.71428571 0.28571429)
        22) Age>=111 14  2 absent (0.85714286 0.14285714) *
        23) Age< 111 7  3 present (0.42857143 0.57142857) *
   3) Start< 8.5 19  8 present (0.42105263 0.57894737) *
```

The rxDTree model suggests the following—for Start < 8.5, 11 of 19 observed subjects developed Kyphosis, while none of the 29 subjects with Start >= 14.5 did. For the remaining 33 subjects, Age was the primary splitting factor; ages 5 to 9 had the highest probability of developing Kyphosis.

The returned object kyphTree is an object of class rxDTree. The rxDTree class, of course, has similar components to an rpart object: frame, cptable, splits, etc. In fact, you can use the rxAddInheritance function to add rpart inheritance to rxDTree objects.

A Simple Regression Tree

As a simple example of a regression tree, consider mtcars, a data set from the 1974 Motor Trend US magazine containing information on fuel consumption, automobile design, and performance. A model fitting gas mileage (mpg) using displacement (disp) as a predictor is specified as follows:

```
mtcarTree <- rxDTree(mpg ~ disp, data=mtcars)
mtcarTree
Call:
rxDTree(formula = mpg ~ disp, data = mtcars)
Data: mtcars
Number of valid observations:  32
Number of missing observations:  0

Tree representation:
n= 32

node), split, n, deviance, yval
      * denotes terminal node

1) root 32 1126.0470 20.09063
  2) disp>=163.5 18  143.5894 15.99444 *
  3) disp< 163.5 14  292.1343 25.35714 *
```

There's a clear split between larger cars (those with engine displacement greater than 163.5 cubic inches) and smaller cars.

Large Data Tree Models

As an example of a large data regression tree, consider the following simple model using a large data set containing information on all of the airline arrivals in the U.S. from the years 1987 to 2008, with over 120 million valid observations, specifying the dependent variable Late as an arrival delay greater than 15 minutes. The default cp of 0 produces a very large number of splits; specifying cp = 1e-5 produces a more manageable set of splits in this model:

```
airlineTree <- rxDTree(Late ~ CRSDepTime + DayOfWeek,
    data=airlineData, blocksPerRead=30, maxDepth=5, cp=1e-5)
airlineTree
Call:
rxDTree(formula = Late ~ CRSDepTime + DayOfWeek, data = airlineData,
    maxDepth = 5, cp = 1e-05, blocksPerRead = 30)
File:  C:\data\AirlineData87to08.xdf
Number of valid observations:  120947440
Number of missing observations:  2587529
```

```
Tree representation:
n= 120947440

node), split, n, deviance, yval
      * denotes terminal node

 1) root 120947440 18861730.00 0.19332390
   2) CRSDepTime< 12.5 56446630  7067701.00 0.14674420
     4) CRSDepTime< 7.5 17283359  1726275.00 0.11254770
       8) CRSDepTime>=0.5 16383303  1571920.00 0.10750350
        16) CRSDepTime< 6.5 7764782   664235.10 0.09446898
          32) CRSDepTime>=1.5 7552115   637099.50 0.09301156 *
          33) CRSDepTime< 1.5 212667    26549.88 0.14622390 *
        17) CRSDepTime>=6.5 8618521   905177.40 0.11924680
          34) DayOfWeek=Sunday 1007650    79145.78 0.08592865 *
          35) DayOfWeek=Monday,Tuesday,Wednesday,Thursday,Friday,Saturday
7610871   824764.90 0.12365800 *
       9) CRSDepTime< 0.5 900056   146349.70 0.20436620 *
     5) CRSDepTime>=7.5 39163271  5312295.00 0.16183560
      10) DayOfWeek=Sunday 5448470   633498.80 0.13431020
        20) CRSDepTime< 10.5 3248411   339124.30 0.11842040
          40) CRSDepTime< 8.5 1149198   109104.30 0.10622280 *
          41) CRSDepTime>=8.5 2099213   229755.40 0.12509780 *
        21) CRSDepTime>=10.5 2200059   292343.30 0.15777170 *
      11) DayOfWeek=Monday,Tuesday,Wednesday,Thursday,Friday,Saturday
33714801  4674001.00 0.16628380
        22) DayOfWeek=Monday,Tuesday,Wednesday,Saturday 22361826
3004645.00 0.15994840
          44) CRSDepTime< 8.5 5009282   626239.90 0.14646910 *
          45) CRSDepTime>=8.5 17352544  2377232.00 0.16383960 *
        23) DayOfWeek=Thursday,Friday 11352975  1666691.00 0.17876260
          46) CRSDepTime< 10.5 6900585   969283.50 0.16903770 *
          47) CRSDepTime>=10.5 4452390   695743.20 0.19383480 *
   3) CRSDepTime>=12.5 64500810 11564380.00 0.23408730
     6) DayOfWeek=Monday,Tuesday,Wednesday,Saturday,Sunday 45661578
7763300.00 0.21718950
      12) DayOfWeek=Saturday 7912641  1158448.00 0.17813770
        24) CRSDepTime>=19.5 1282735   174018.60 0.16186120 *
        25) CRSDepTime< 19.5 6629906   984023.60 0.18128690 *
      13) DayOfWeek=Monday,Tuesday,Wednesday,Sunday 37748937  6590255.00
0.22537520
        26) CRSDepTime< 15.5 13018323  2067330.00 0.19800920
          52) CRSDepTime< 14.5 8711520  1338418.00 0.18957720 *
          53) CRSDepTime>=14.5 4306803   727039.90 0.21506490 *
        27) CRSDepTime>=15.5 24730614  4508044.00 0.23978080
          54) DayOfWeek=Monday,Tuesday 12361529  2159837.00 0.22563250 *
          55) DayOfWeek=Wednesday,Sunday 12369085  2343259.00 0.25392040 *
     7) DayOfWeek=Thursday,Friday 18839232  3756440.00 0.27504340
      14) CRSDepTime< 15.5 6512591  1184197.00 0.23891000
        28) CRSDepTime< 14.5 4360337   767152.30 0.22785810
          56) DayOfWeek=Thursday 2177374   372175.90 0.21880390 *
          57) DayOfWeek=Friday 2182963   394619.90 0.23688900 *
        29) CRSDepTime>=14.5 2152254   415433.50 0.26130050
          58) DayOfWeek=Thursday 1074247   201968.30 0.25102050 *
          59) DayOfWeek=Friday 1078007   213238.60 0.27154460 *
      15) CRSDepTime>=15.5 12326641  2559247.00 0.29413390
        30) CRSDepTime>=21.5 769409   142209.10 0.24471380 *
        31) CRSDepTime< 21.5 11557232  2415033.00 0.29742400
          62) CRSDepTime< 16.5 2132965   424986.90 0.27471570 *
          63) CRSDepTime>=16.5 9424267  1988698.00 0.30256350 *
```


Looking at the fitted objects cptable component, we can look at whether we have overfitted the model:

```
airlineCTree$cptable
              CP nsplit rel error    xerror         xstd
1  1.217538e-02      0 1.0000000 1.0000000 0.0001412275
2  2.366644e-03      1 0.9878246 0.9878246 0.0001391320
3  1.544427e-03      2 0.9854580 0.9854580 0.0001389551
4  7.814231e-04      3 0.9839136 0.9839136 0.0001385057
5  6.889827e-04      5 0.9823507 0.9823508 0.0001383059
6  4.243991e-04      6 0.9816617 0.9816619 0.0001382856
7  2.622974e-04      7 0.9812373 0.9812376 0.0001381622
8  2.542254e-04      8 0.9809750 0.9809753 0.0001381414
9  1.413135e-04      9 0.9807208 0.9807211 0.0001380756
10 1.329552e-04     10 0.9805795 0.9805836 0.0001380441
11 1.076881e-04     11 0.9804465 0.9804506 0.0001379937
12 1.062610e-04     12 0.9803388 0.9803455 0.0001379770
13 9.925921e-05     13 0.9802326 0.9802367 0.0001379596
14 8.822443e-05     14 0.9801333 0.9800887 0.0001379346
15 8.544236e-05     15 0.9800451 0.9800887 0.0001379346
16 7.151164e-05     16 0.9799597 0.9799579 0.0001379193
17 6.715654e-05     17 0.9798881 0.9798864 0.0001379197
18 6.218350e-05     18 0.9798210 0.9798193 0.0001378946
19 3.105152e-05     19 0.9797588 0.9797612 0.0001378802
20 2.150280e-05     20 0.9797278 0.9797301 0.0001378687
21 1.890252e-05     21 0.9797063 0.9797086 0.0001378641
22 1.402735e-05     22 0.9796873 0.9796897 0.0001378622
23 1.201657e-05     23 0.9796733 0.9796757 0.0001378573
24 1.000000e-05     24 0.9796613 0.9796637 0.0001378568
```

We see a steady decrease in cross-validation error (`xerror`) as the number of splits increase, but note that at about `nsplit=11` the rate of change slows dramatically. The optimal model is probably very near here.

To prune the tree back, we can use the `prune.rxDTree` function:

```
airlineTree4 <- prune.rxDTree(airlineTree, cp=1e-4)
airlineTree4
Call:
rxDTree(formula = Late ~ CRSDepTime + DayOfWeek, data = airlineData,
    maxDepth = 5, cp = 1e-05, blocksPerRead = 30)
File:  C:\data\AirlineData87to08.xdf
Number of valid observations:  120947440
Number of missing observations:  2587529

Tree representation:
n= 120947440

node), split, n, deviance, yval
      * denotes terminal node

 1) root 120947440 18861730.0 0.19332390
   2) CRSDepTime< 12.5 56446630  7067701.0 0.14674420
     4) CRSDepTime< 7.5 17283359  1726275.0 0.11254770
       8) CRSDepTime>=0.5 16383303  1571920.0 0.10750350
        16) CRSDepTime< 6.5 7764782   664235.1 0.09446898 *
        17) CRSDepTime>=6.5 8618521   905177.4 0.11924680 *
       9) CRSDepTime< 0.5 900056   146349.7 0.20436620 *
     5) CRSDepTime>=7.5 39163271  5312295.0 0.16183560
```

```
      10) DayOfWeek=Sunday 5448470   633498.8 0.13431020
        20) CRSDepTime< 10.5 3248411   339124.3 0.11842040 *
        21) CRSDepTime>=10.5 2200059   292343.3 0.15777170 *
      11) DayOfWeek=Monday,Tuesday,Wednesday,Thursday,Friday,Saturday
33714801  4674001.0 0.16628380
        22) DayOfWeek=Monday,Tuesday,Wednesday,Saturday 22361826
3004645.0 0.15994840 *
        23) DayOfWeek=Thursday,Friday 11352975  1666691.0 0.17876260 *
   3) CRSDepTime>=12.5 64500810 11564380.0 0.23408730
     6) DayOfWeek=Monday,Tuesday,Wednesday,Saturday,Sunday 45661578
7763300.0 0.21718950
      12) DayOfWeek=Saturday 7912641  1158448.0 0.17813770 *
      13) DayOfWeek=Monday,Tuesday,Wednesday,Sunday 37748937  6590255.0
0.22537520
        26) CRSDepTime< 15.5 13018323  2067330.0 0.19800920 *
        27) CRSDepTime>=15.5 24730614  4508044.0 0.23978080
          54) DayOfWeek=Monday,Tuesday 12361529  2159837.0 0.22563250 *
          55) DayOfWeek=Wednesday,Sunday 12369085  2343259.0 0.25392040 *
     7) DayOfWeek=Thursday,Friday 18839232  3756440.0 0.27504340
      14) CRSDepTime< 15.5 6512591  1184197.0 0.23891000 *
      15) CRSDepTime>=15.5 12326641  2559247.0 0.29413390
        30) CRSDepTime>=21.5 769409   142209.1 0.24471380 *
        31) CRSDepTime< 21.5 11557232  2415033.0 0.29742400 *
```

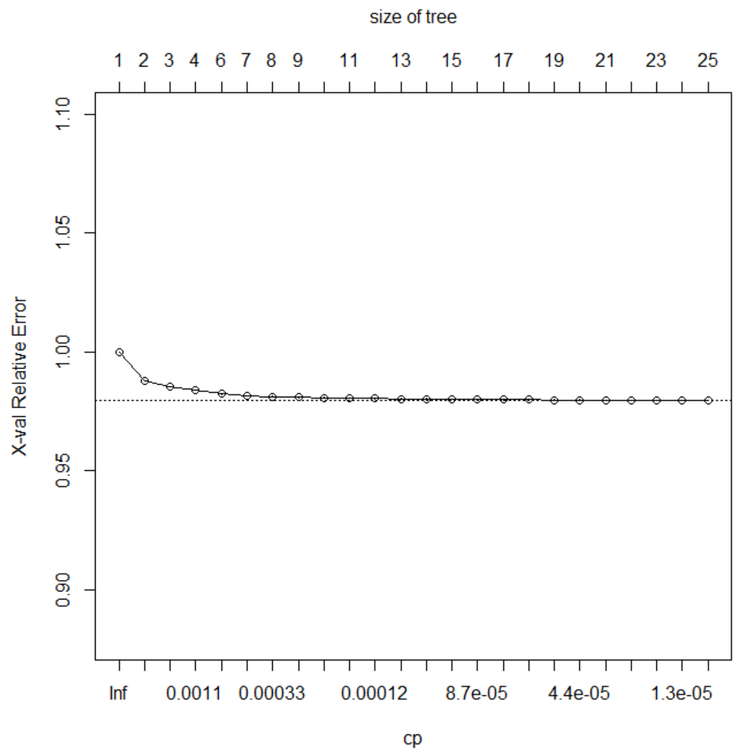
If the `rpart` package is loaded, `prune.rxDTree` acts as a method for the prune function, so you can call it more simply:

```
airlineTree4 <- prune(airlineTree, cp=1e-4)
```

For models fit with 2-fold or greater cross-validation, it is useful to use the cross-validation standard error (part of the `cptable` component) as a guide to pruning. The `rpart` function `plotcp` can be useful for this:

```
plotcp(rxAddInheritance(airlineCTree))
```

This yields the following plot:



From this plot, it appears we can prune even further, to perhaps seven or eight splits. Looking again at the `cptable`, a cp of 2.5e-4 seems a reasonable pruning choice:

```
airlineTreePruned <- prune.rxDTree(airlineTree, cp=2.5e-4)
airlineTreePruned
Call:
rxDTree(formula = Late ~ CRSDepTime + DayOfWeek, data = airlineData,
    maxDepth = 5, cp = 1e-05, blocksPerRead = 30)
File:  C:\data\AirlineData87to08.xdf
Number of valid observations:  120947440
Number of missing observations:  2587529

Tree representation:
n= 120947440

node), split, n, deviance, yval
      * denotes terminal node

 1) root 120947440 18861730.0 0.1933239
   2) CRSDepTime< 12.5 56446630  7067701.0 0.1467442
     4) CRSDepTime< 7.5 17283359  1726275.0 0.1125477
       8) CRSDepTime>=0.5 16383303  1571920.0 0.1075035 *
       9) CRSDepTime< 0.5 900056   146349.7 0.2043662 *
     5) CRSDepTime>=7.5 39163271  5312295.0 0.1618356
      10) DayOfWeek=Sunday 5448470   633498.8 0.1343102 *
      11) DayOfWeek=Monday,Tuesday,Wednesday,Thursday,Friday,Saturday
33714801  4674001.0 0.1662838 *
   3) CRSDepTime>=12.5 64500810 11564380.0 0.2340873
     6) DayOfWeek=Monday,Tuesday,Wednesday,Saturday,Sunday 45661578
7763300.0 0.2171895
      12) DayOfWeek=Saturday 7912641  1158448.0 0.1781377 *
```

```
      13) DayOfWeek=Monday,Tuesday,Wednesday,Sunday 37748937  6590255.0 
0.2253752  
        26) CRSDepTime< 15.5 13018323  2067330.0 0.1980092 *
        27) CRSDepTime>=15.5 24730614  4508044.0 0.2397808  
          54) DayOfWeek=Monday,Tuesday 12361529  2159837.0 0.2256325 *
          55) DayOfWeek=Wednesday,Sunday 12369085  2343259.0 0.2539204 *
     7) DayOfWeek=Thursday,Friday 18839232  3756440.0 0.2750434  
      14) CRSDepTime< 15.5 6512591  1184197.0 0.2389100 *
      15) CRSDepTime>=15.5 12326641  2559247.0 0.2941339 *
```

Controlling the Model Fit

The `rxDTree` function has a number of options for controlling the model fit. These allow you to control such things as the complexity parameter, the number of folds used to perform cross-validation, the depth of the tree, and the size of terminal nodes. You can also control the number of bins used by the `rxDTree` algorithm.

Handling Missing Values

The `removeMissings` argument to `rxDTree`, as in most RevoScaleR analysis functions, controls how the function deals with missing data in the model fit. If TRUE, all rows containing missing values for the response or any predictor variable are removed before model fitting. If FALSE (the default), only those rows for which either the response or all values of predictor variables are missing are removed.

Prediction

As with other RevoScaleR analysis functions, prediction is performed using the `rxPredict` function, to which you supply a fitted model object and a set of new data (which may be the original data set, but in any event must contain the variables used in the original model).

The adult data set (Kohavi, 1996) is a widely used machine learning data set containing information on adult workers. The data set is available from the machine learning data repository at UC Irvine ((Frank & Asuncion, 2010) and comes in two pieces: a training data set (`adult.data`) and a test data set (`adult.test`). This makes it ready-made for use in prediction. (A third file, `adult.names`, gives a description of the variables; we use this in the code below as a source for the variable names, which are not part of the data files):

```
adult.train <- read.table("C:/data/adult.data", sep=",", 
    stringsAsFactors=TRUE)
names(adult.train) <- c("age", "workclass", "fnlwgt", "education", 
        "education_num", "marital_status", "occupation", 
        "relationship", "race", "sex", "capital_gain", 
        "capital_loss", "hours_per_week",     "native_country", 
        "income")
adult.test <- read.table("C:/data/adult.test", skip=1, sep=",", 
    stringsAsFactors=TRUE)
names(adult.test) <- c("age", "workclass", "fnlwgt", "education", 
        "education_num", "marital_status", "occupation", 
        "relationship","race", "sex", "capital_gain", "capital_loss", 
        "hours_per_week",      "native_country", "income")

adult.tree <- rxDTree(income ~ age + sex + hours_per_week, 
        pweights="fnlwgt", data=adult.train)
```

```
adult.pred <- rxPredict(adult.tree, data=adult.test,
       type="vector")

mean(adult.pred == as.integer(adult.test$income))
[1] 0.7734169
```

The result shows that the fitted model accurately classifies about 77% of the test data.

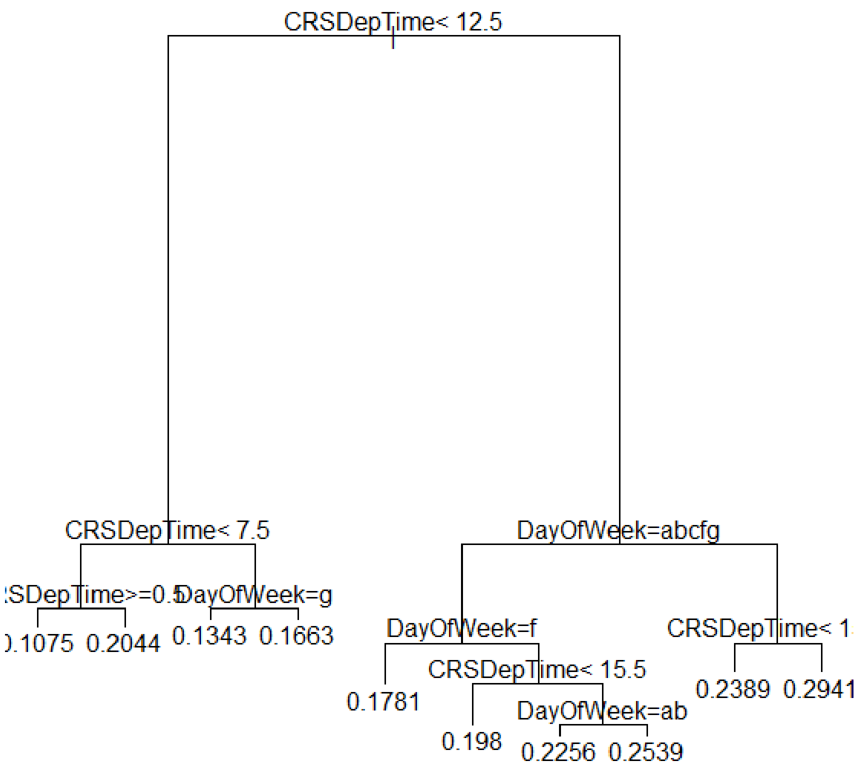
When using `rxPredict` with `rxDTree` objects, you should keep in mind how it differs from predict with `rpart` objects. First, a data argument is always required—this can be either the original data or new data; there is no new`data` argument as in `predict.rpart`. Prediction with the original data provides fitted values, not predictions, but the predicted variable name still defaults to `varname_Pred`.

Plotting Trees

You can use the `rpart` plot and text methods with `rxDTree` objects, provided you use the `rxAddInheritance` function to provide `rpart` inheritance:

```
plot(rxAddInheritance(airlineCTreePruned))
text(rxAddInheritance(airlineCTreePruned))
```

This provides the following plot:



References

Ben-Haim, Y., & Tom-Tov, E. (2010). A streaming parallel decision tree algorithm. Journal of Machine Learning Research, 849-872.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). Classification and Regression Trees. Pacific Grove: Wadsworth.

Frank, A., & Asuncion, A. (2010). (University of California, Irvine, School of Information and Computer Science) Retrieved August 2012, from UCI Machine Learning Repository: http://archive.ics.uci.edu/ml

Kohavi, R. (1996). Scaling Up the Accuracy of Naive-Bayes Classifiers: a Decision-Tree Hybrid. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining.

Therneau, T., & Atkinson, E. (1997). An Introduction to Recursive Partitioning Using the RPART Routines. Rochester, MN: Mayo Clinic.

About Revolution Analytics

Revolution Analytics is the leading commercial provider of software and services based on the open source R project for statistical computing. Led by predictive analytics pioneer Norman Nie, the company brings high performance, productivity and enterprise readiness to R, the most powerful statistics language in the world. The company's flagship Revolution R Enterprise product is designed to meet the production needs of large organizations in industries such as finance, life sciences, retail, manufacturing and media.

Please visit us at www.revolutionanalytics.com