CRTREES: AN IMPLEMENTATION OF CLASSIFICATION AND REGRESSION TREES (CART) & RANDOM FORESTS IN STATA

Ricardo Mora

Universidad Carlos III de Madrid

Madrid, October 2019

Outline

Introduction

Decision trees

- Decision tree-structured models are predictive models that use tree-like diagrams
  - Classification trees: the target variable takes a finite set of values
  - Regression trees: the target variable takes real numbers

Decision trees

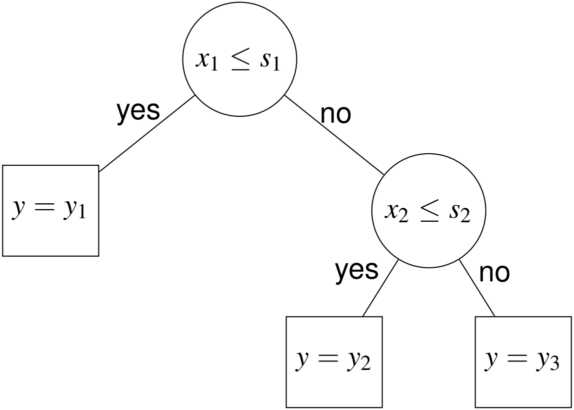
- Decision tree-structured models are predictive models that use tree-like diagrams
  - Classification trees: the target variable takes a finite set of values
  - Regression trees: the target variable takes real numbers
- Each branch in the tree represents a sample split criterion

Decision trees

- Decision tree-structured models are predictive models that use tree-like diagrams
  - Classification trees: the target variable takes a finite set of values
  - Regression trees: the target variable takes real numbers
- Each branch in the tree represents a sample split criterion
- Several Approaches:
  - Chi-square automated interaction detection, CHAID (Kass 1980; Biggs et al. 1991)
  - Classification and Regression Trees, CART (Breiman et al. 1984)
  - Random Forests (Breiman 2001; Scornet et al. 2015)

A simple tree structure

$$y(x_1, x_2) \begin{cases} = y_1 \text{ if } x_1 \leq s_1 \\ = y_2 \text{ if } x_1 > s_1 \text{ and } x_2 \leq s_2 \\ = y_3 \text{ if } x_1 > s_1 \text{ and } x_2 > s_2 \end{cases}$$



CART

- CART objective is to estimate a binary tree structure

CART

- CART objective is to estimate a binary tree structure
- It performs three algorithms:
  - *Tree-growing*: step-optimal recursive partition (LS on 50 cells with at most two terminal nodes $\approx 6 \times 10^{14}$ models)
  - *Tree-pruning* & *Obtaining the honest tree*

CART

- CART objective is to estimate a binary tree structure
- It performs three algorithms:
  - *Tree-growing*: step-optimal recursive partition (LS on 50 cells with at most two terminal nodes $\approx 6 \times 10^{14}$ models)
  - *Tree-pruning* & *Obtaining the honest tree*
- The last two algorithms attempt to minimize *overfitting* (growing trees with no external validity)
  - test sample
  - cross-validation, bootstrap

CART

- CART objective is to estimate a binary tree structure
- It performs three algorithms:
  - *Tree-growing*: step-optimal recursive partition (LS on 50 cells with at most two terminal nodes $\approx 6 \times 10^{14}$ models)
  - *Tree-pruning* & *Obtaining the honest tree*
- The last two algorithms attempt to minimize *overfitting* (growing trees with no external validity)
  - test sample
  - cross-validation, bootstrap
- In Stata, modules <`chaid`> perform CHAID and <`cart`> performs CART analysis for failure time data

Random Forests

- Random forests is an *ensemble learning* method to generate predictions using tree structures

Random Forests

- Random forests is an *ensemble learning* method to generate predictions using tree structures

- *Ensemble learning* method: use of many strategically generated models
  - First step: create multitude of (presumably over-fitted) trees with tree-growing algorithm
    - The multitude of trees are obtained by random sampling (bagging) and by random choice of splitting variables
  - Second step: case predictions are built using modes (in classification) and averages (in regression)

Random Forests

- Random forests is an *ensemble learning* method to generate predictions using tree structures

- *Ensemble learning* method: use of many strategically generated models
  - First step: create multitude of (presumably over-fitted) trees with tree-growing algorithm
    - The multitude of trees are obtained by random sampling (bagging) and by random choice of splitting variables
  - Second step: case predictions are built using modes (in classification) and averages (in regression)

- In Stata, <sctree> is a Stata wrapper for the R functions "tree()", "randomForest()", and "gbm()"
  - Classification tree with optimal pruning, bagging, boosting, and random forests

Algorithms

Growing the tree (CART & Random Forests)

Growing the tree (CART & Random Forests)

- Requires a so-called *training* or *learning sample*

Growing the tree (CART & Random Forests)

- Requires a so-called *training* or *learning sample*
- At iteration i with tree structure T_i consider all terminal nodes $t^*(T_i)$
  - Classification: Let $i(T_i)$ be an overall impurity measure (using the gini or entropy index)
  - Regression: Let $i(T_i)$ be the residual sum of squares in all terminal nodes
  - The best split at iteration i identifies the terminal node and split criterion that maximizes $i(T_i) - i(T_{i+1})$

Growing the tree (CART & Random Forests)

- Requires a so-called *training* or *learning sample*
- At iteration i with tree structure T_i consider all terminal nodes $t^*(T_i)$
  - Classification: Let $i(T_i)$ be an overall impurity measure (using the gini or entropy index)
  - Regression: Let $i(T_i)$ be the residual sum of squares in all terminal nodes
  - The best split at iteration i identifies the terminal node and split criterion that maximizes $i(T_i) - i(T_{i+1})$
- *Recursive partitioning* ends with the largest possible tree, T_{MAX} where there are no nodes to split or the number of observations reach a lower limit (splitting rule)

Overfitting and aggregation bias

Overfitting and aggregation bias

- In a trivial setting, the result is equivalent to dividing the sample into all possible cells and computing within-cell least squares

Overfitting and aggregation bias

- In a trivial setting, the result is equivalent to dividing the sample into all possible cells and computing within-cell least squares
- Overfitting: T_{MAX} will usually be too complex in the sense that it has no external validity and some terminal nodes should be aggregated
  - Besides, a more simplified structure will normally lead to more accurate estimates since the number of observations in each terminal node grows as aggregation takes place

Overfitting and aggregation bias

- In a trivial setting, the result is equivalent to dividing the sample into all possible cells and computing within-cell least squares
- Overfitting: T_{MAX} will usually be too complex in the sense that it has no external validity and some terminal nodes should be aggregated
  - Besides, a more simplified structure will normally lead to more accurate estimates since the number of observations in each terminal node grows as aggregation takes place
- However, if aggregation goes too far, aggregation bias becomes a serious problem

Pruning the tree: Error-complexity clustering (CART)

- In order to avoid overfitting, CART identifies a sequence of nested trees that results from recursive aggregation of nodes from T_{MAX} with a clustering procedure

Pruning the tree: Error-complexity clustering (CART)

- In order to avoid overfitting, CART identifies a sequence of nested trees that results from recursive aggregation of nodes from T_{MAX} with a clustering procedure
- For a given value α, let $R(\alpha, T) = R(T) + \alpha |T|$ where $|T|$ denotes the number of terminal nodes, or complexity, of tree T and $R(T)$ is the MSE in regression or the misclassification rate in classification

Pruning the tree: Error-complexity clustering (CART)

- In order to avoid overfitting, CART identifies a sequence of nested trees that results from recursive aggregation of nodes from T_{MAX} with a clustering procedure

- For a given value α, let $R(\alpha, T) = R(T) + \alpha |T|$ where $|T|$ denotes the number of terminal nodes, or complexity, of tree T and $R(T)$ is the MSE in regression or the misclassification rate in classification

- The optimal tree for a given α, $T(\alpha)$, minimizes $R(\alpha, T)$ within the set of subtrees of T_{MAX}
    - $T(\alpha)$ belongs to a much broader set than the sequence of trees obtained in the growing algorithm

Pruning the tree: Error-complexity clustering (CART)

- In order to avoid overfitting, CART identifies a sequence of nested trees that results from recursive aggregation of nodes from T_{MAX} with a clustering procedure
- For a given value α, let $R(\alpha, T) = R(T) + \alpha |T|$ where $|T|$ denotes the number of terminal nodes, or complexity, of tree T and $R(T)$ is the MSE in regression or the misclassification rate in classification
- The optimal tree for a given α, $T(\alpha)$, minimizes $R(\alpha, T)$ within the set of subtrees of T_{MAX}
  - $T(\alpha)$ belongs to a much broader set than the sequence of trees obtained in the growing algorithm
- *Pruning* identifies a sequence of real positive numbers $\{\alpha_0, \alpha_1, ..., \alpha_M\}$ such that $\alpha_j < \alpha_{j+1}$ and $T_{MAX} \equiv T(\alpha_0) \succ T(\alpha_1) \succ T(\alpha_2) \succ \ldots \succ \{\text{root}\}$

Honest tree (CART)

- Out of the sequence of optimal trees, $\{T(\alpha_j)\}_j$, T_{MAX} has lowest $R(T)$ in the learning sample by construction and $R(\cdot)$ increases with α

Honest tree (CART)

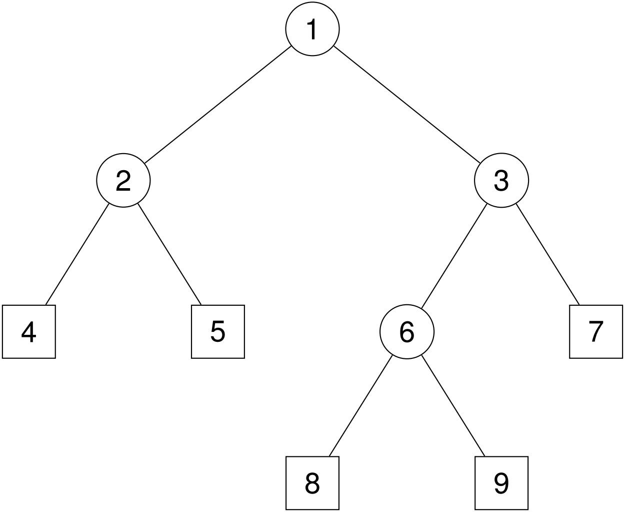
- Out of the sequence of optimal trees, $\{T(\alpha_j)\}_j$, T_{MAX} has lowest $R(T)$ in the learning sample by construction and $R(\cdot)$ increases with α

- The *honest tree* algorithm chooses the simplest tree that minimizes

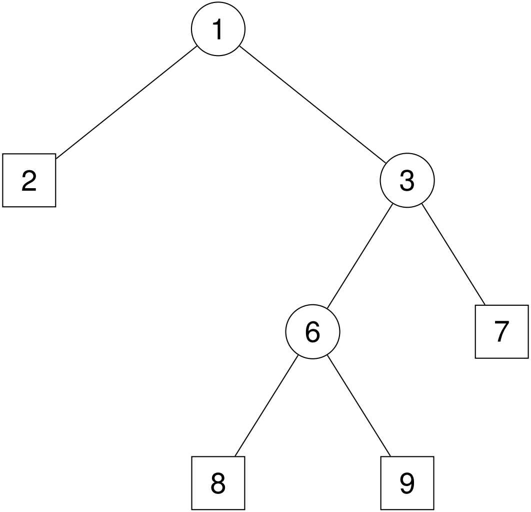
$$R(T) + s \times \text{SE}(R(T)), \qquad s \geq 0$$

  - With partitioning into a learning and a test sample, $R(T)$ and $\text{SE}(R(T))$ are obtained using the test sample
  - With V-fold cross validation the sample is randomly partitioned V times into a learning and a test sample. For each α_j, $R(T)$ and $\text{SE}(R(T))$ are obtained through averaging of results in the V partitions
  - With the bootstrap under regression, $s > 0$ and $\text{SE}(R(T_{MAX}))$ is obtained using the bootstrap

T_{MAX} : 5 terminal nodes



T_1: Node 2 becomes terminal



T_2: Node 1 becomes terminal

$$\boxed{1}$$

T_2: Node 1 becomes terminal

$$\boxed{1}$$

- The sequence of optimal trees is

$$\{T_{MAX}, T_1, T_2 \equiv \{\text{root}\}\}$$

with $|T_{MAX}| = 5$, $|T_1| = 4$, $|T_2| = 1$

T_2: Node 1 becomes terminal

$\boxed{1}$

- The sequence of optimal trees is

$$\{T_{MAX}, T_1, T_2 \equiv \{\text{root}\}\}$$

  with $|T_{MAX}| = 5$, $|T_1| = 4$, $|T_2| = 1$

- Using a test sample, among the three we would choose the tree that gives a smaller $R^{ts}(T) + s \times \text{SE}(R^{ts}(T))$

CART properties

- Some basic results for recursive partitioning can be found in (Breiman et al. 1984, ch.12)

CART properties

- Some basic results for recursive partitioning can be found in (Breiman et al. 1984, ch.12)
- Consistency requires an ever more dense sample at all n-dimensional balls of the input space

CART properties

- Some basic results for recursive partitioning can be found in (Breiman et al. 1984, ch.12)
- Consistency requires an ever more dense sample at all n-dimensional balls of the input space
- Cost-complexity minimization together with test-sample $R(\cdot)$ should help this condition is not to strong

CART properties

- Some basic results for recursive partitioning can be found in (Breiman et al. 1984, ch.12)
- Consistency requires an ever more dense sample at all n-dimensional balls of the input space
- Cost-complexity minimization together with test-sample $R(\cdot)$ should help this condition is not to strong
- For small samples correlation in splitting variables
  - induces instability in the tree topology
  - interpretation of the contribution of each splitting variable is problematic

Bagging (Random Forests)

Bagging (Random Forests)

- *Bagging* is an ensemble method that reduces the problem of overfitting by trading off large bias in each model considered with higher accuracy and less bias by aggregating results from all models considered
    - **b**ootstrapping to generate a multitude of models
    - **agg**regat**ing** to make a final prediction (mode in classification; average in regression)

Bagging (Random Forests)

- *Bagging* is an ensemble method that reduces the problem of overfitting by trading off large bias in each model considered with higher accuracy and less bias by aggregating results from all models considered
  - **b**ootstrapping to generate a multitude of models
  - **agg**regat**ing** to make a final prediction (mode in classification; average in regression)

- Two methods to simultaneously obtain alternative models:
  - Sampling observations
  - Sampling splitting variables

Bagging (Random Forests)

- *Bagging* is an ensemble method that reduces the problem of overfitting by trading off large bias in each model considered with higher accuracy and less bias by aggregating results from all models considered
  - **b**ootstrapping to generate a multitude of models
  - **agg**regat**ing** to make a final prediction (mode in classification; average in regression)

- Two methods to simultaneously obtain alternative models:
  - Sampling observations
  - Sampling splitting variables

- Focus in Random Forests is on prediction, not interpretation

Large sample properties

- Breiman et al. (1984): Consistency in recursive splitting algorithms
- Sexton and Laake (2009): Jackknife standard error estimator in bagged ensemble
- Mentch and Hooker (2014): Asymptotic sampling distribution in Random Forests
- Efron (2014): Estimators for standard errors for the predictions in bagged Random Forests (Infinitesimal Jackknife and the Jackknife-after-Bootstrap)
- Scornet et al. (2015): First consistency result for the original Breiman (2001) algorithm in the context of regression models

crtrees

The crtrees ado

`crtrees` *depvar varlist* [*if*] [*in*], `options`

- *depvar*: output variable (discrete in classification)
- *varlist*: splitting variables (binary, ordinal, or cardinal)
- the command implements both CART and Random Forests in classification and regression problems
- by default, the command performs Regression Trees (CART in a regression problem) with a constant in each terminal node using a test sample with 50 percent the original sample size, and the 0 SE rule for estimating the honest tree

Model Options

- **rfor**ests: performs growing the tree and bagging. By default, crtrees performs CART
- **class**ification: performs classification trees
- **gen**erate(*newvar*): new variable name for model predictions. This is required when options st_code and/or rforests are used
- **boot**straps(#): only available for regression trees (to obtain SE (T_{MAX})) and for rforests (for bagging)
- **seed**(#), **stop**(#): seed for random number generator and stopping rule for growing the tree

Options for regression problems (both in CART and Random Forests)

- **reg**`ressors(`*`varlist`*`)`: controls in terminal nodes. A regression line is estimated in each terminal node
- **nocon**`stant`: regression line does not include the constant
- **`level(#)`**: sets confidence level for regression output display when test sample is used (this option is available with CART)

Options for classification problems (both in CART and Random Forests)

- `impurity(string)`: impurity measure, either “gini” or “entropy”
- `priors(string)`: Stata matrix with prior class probabilities (learning sample frequencies by default)
- `costs(string)`: name of Stata matrix with costs of misclassification. By default, they are 0 in diagonal and 1 elsewhere
- `detail`: displays additional statistics for terminal nodes

CART options

- **<u>lss</u>**`ize(#)`: proportion of the learning sample (default is 0.5)
- **<u>ts</u>**`ample(`*`newvar`*`)`: identifies test sample observations (e(sample) includes also the learning sample)
- **<u>vcv</u>**`(#)`: sets V-fold cross validation parameter
- **<u>rule</u>**`(#)`: SE rule to identify honest tree
- **<u>t</u>**`ree`: text representation of estimated tree
- **<u>st</u>**`_code`: Stata code to generate tree predictions

Random Forests options

- **rsplit**`ting(#)`: relative size for subsample splitting variables (default is 0.33)
- **rsampl**`ing(#)`: relative subsample size (default is 1, with replacement; otherwise, without replacement)
- **oob**: out-of-bag misclassification costs using observations not included in their bootstrap sample (default is using all observations)
- **ij**: standard errors using Infinitessimal Jacknife (the nonparametric delta method); only available with regression problems; default is jackknife-after-bootstrap
- **save**`trees(`*string*`)`: name of file to save mata matrices from multitude of trees
  - this is required to run `predict` after `crtrees` with `rforests` option. No automatic replacement of existing file is allowed. If unspecified, `crtrees` will save in the current working directory the file `matatrees`

crtrees_p

After `crtrees`, we can use `predict` to obtain model predictions in the same or alternative samples

- the model predictions are computed using the honest tree under CART, the average prediction of all trees from *bagging* with `rforest` in a regression problem, or the most popular vote from all trees from *bagging* with `rforests` in a classification problem
- with `rforest` in a regression problem, it also creates a new variable with the standard error of the prediction using all trees from *bagging*
- with `rforests` in a classification problem, it also creates a new variable containing the bootstrap misclassification cost (by default, the probability of misclassification) using all trees from *bagging*

Examples with auto data

Regression trees without controls

```
crtrees price trunk weight length foreign gear_ratio,
                    seed(12345) rule(2)
```

- regression trees with sample partition, learning sample 0.5 and 2 SE rule
- the seed is required to ensure replicability because partitioning the sample is random

Regression trees without controls (cont'd)

```
Regression Trees with learning and test samples (SE rule: 2)
Learning Sample                                      Test Sample
|T*|           = 2
Number of obs = 37                                   Number of obs    =        37
R-squared     = 0.5330                               R-squared        =    0.3769
Avg Dep Var   = 6205.378                             Avg Dep Var      =  6125.135
Root MSE      = 2133.378                             Root MSE         =  2287.073
Terminal node results:

Node 2:
     Characteristics:
        1760<=weight<=3740
        2.24<=gear_ratio<=3.89
     Number of obs =        32
     Average       =  5329.125
     Std.Err.      =     329.8
Node 3:
     Characteristics:
        3830<=weight<=4840
        149<=length<=233
        2.19<=gear_ratio<=3.81
     Number of obs =         5
     Average       =   11813.4
     Std.Err.      =      1582
```

Regression trees with controls

```
crtrees price trunk weight length foreign gear_ratio,
   reg(weight) stop(5) lssize(0.6) generate(y_hat)
                  seed(12345) rule(1)
```

- variable `weight` is both splitting variable and control
- growing the tree stops when the regression cannot be computed or when the number of observations is smaller or equal to 5
- new variable `y_hat` includes predictions

Regression trees with controls (cont'd)

```
Regression Trees with learning and test samples (SE rule: 1)

Learning Sample                                    Test Sample
|T*|          = 2
Number of obs = 44                                 Number of obs    =        30
R-squared     = 0.5814                             R-squared        =    0.4423
Avg Dep Var   = 6175.091                           Avg Dep Var      =  6150.833
Root MSE      = 2008.796                           Root MSE         =  2258.638

Terminal node results:

Node 2:
     Characteristics:
        147<=length<=233
        foreign==0
        2.19<=gear_ratio<=3.81
     Number of obs =          29
                                                   R-squared        =    0.4900
```

price	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
weight	3.185787	.6643858	4.80	0.000	1.883614	4.487959
_const	-4520.597	2219.4	-2.04	0.042	-8870.54	-170.653

```
Node 3:
     Characteristics:
        foreign==1
        2.24<=gear_ratio<=3.89
     Number of obs =          15
                                                   R-squared        =    0.7650
```

price	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
weight	5.277319	.607164	8.69	0.000	4.0873	6.467339
_const	-5702.361	1452.715	-3.93	0.000	-8549.629	-2855.092

Classification trees with V-fold cross-validation

```
crtrees foreign price trunk, class stop(10) vcv(20)
        seed(12345) detail tree rule(0.5)
```

- en each partition, 100/20=5 percent of the sample is test sample
- additional information is presented in the terminal nodes
- tree text representation is displayed

Classification trees with V-fold cross-validation (cont'd)

```
Classification Trees with V-fold Cross Validation (SE rule: .5)
Impurity measure: Gini
Sample                                                V-fold cross validation
Number of obs = 74                                   V                =        20
|T*|          = 3
R(T*)         = 0.1622                               R(T*)            =    0.2472
                                                     SE(R(T*))        =    0.1104
Text representation of tree:
At node 1 if trunk <= 15.5 go to node 2 else go to node 3
At node 2 if price <= 5006.5 go to node 4 else go to node 5
Terminal node results:
Node 3:
     Characteristics:
        16<=trunk<=23
     Class predictor =          0
     r(t)            =      0.065
     Number of obs   =         31
     Pr(foreign=0)   =      0.935
     Pr(foreign=1)   =      0.065

Node 4:
     Characteristics:
        3291<=price<=4934
        5<=trunk<=15
     Class predictor =          0
     r(t)            =      0.259
     Number of obs   =         27
     Pr(foreign=0)   =      0.741
     Pr(foreign=1)   =      0.259

Node 5:
     Characteristics:
        5079<=price<=15906
        5<=trunk<=15
     Class predictor =          1
     r(t)            =      0.188
     Number of obs   =         16
     Pr(foreign=0)   =      0.188
     Pr(foreign=1)   =      0.812
```

Automatic generation of Stata code

```
crtrees foreign price trunk, class stop(10) vcv(20)
seed(12345) detail tree rule(0.5) st_code gen(pr_class)
```

- options `generate()` and `st_code` are required
- in the output display, we can find Stata code lines to generate predictions
- this code can be copied and pasted into do files or can be used as guidance to generate code in other software

Automatic generation of Stata code (cont'd)

```
Classification Trees with V-fold Cross Validation (SE rule: .5)
Impurity measure: Gini
Sample                                              V-fold cross validation
Number of obs = 74                             V                  =        20
|T*|          = 3
R(T*)         = 0.1622                         R(T*)              =    0.2472
                                               SE(R(T*))          =    0.1104
Text representation of tree:
At node 1 if trunk <= 15.5 go to node 2 else go to node 3
At node 2 if price <= 5006.5 go to node 4 else go to node 5
Terminal node results:
Node 3:
     Characteristics:
        16<=trunk<=23
     Class predictor =         0
     r(t)            =     0.065
     Number of obs   =        31
     Pr(foreign=0)   =     0.935
     Pr(foreign=1)   =     0.065

Node 4:
     Characteristics:
        3291<=price<=4934
        5<=trunk<=15
     Class predictor =         0
     r(t)            =     0.259
     Number of obs   =        27
     Pr(foreign=0)   =     0.741
     Pr(foreign=1)   =     0.259

Node 5:
     Characteristics:
        5079<=price<=15906
        5<=trunk<=15
     Class predictor =         1
     r(t)            =     0.188
     Number of obs   =        16
     Pr(foreign=0)   =     0.188
     Pr(foreign=1)   =     0.812

// Stata code to generate predictions
generate pr_class=.
replace pr_class=0 if  3291<=price & price<=15906 &  16<=trunk & trunk<=23
replace pr_class=0 if  3291<=price & price<=4934 &  5<=trunk & trunk<=15
replace pr_class=1 if  5079<=price & price<=15906 &  5<=trunk & trunk<=15
// end of Stata code to generate predictions
```

Random forests with regression

```
crtrees price trunk weight, rforests regressors(weight)
          generate(p_hat) bootstraps(500)
```

- Random Forests requires options `rforests`, `generate`, and `bootstraps`
- subsampling and random selection of splitting variables is controlled with options `rsampling` and `rsplitting`

Random forests with regression (cont'd)

```
Random Forests: Regression
Bootstrap replications (550)
----+--- 100 ---+--- 200 ---+--- 300 ---+--- 400 ---+--- 500
..................................................     500
.....
Dep. Variable = price
Splitting Variables = trunk weight
Regressors  = weight
Bootstraps  = 550
                                          Number of obs     =         74
                                          R-squared         =     0.6079
                                          Model root SS     =      19649
                                          Residual root SS  =      16098
                                          Total root SS     =      25201

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
       p_hat |         74    5954.731    2299.715  -2284.176   12357.13
    p_hat_se |         74    2418.164    3974.634   346.2865   31753.67
Jacknife-after-Bootstrap Standard Errors
(Note: computing time: 4.62 seconds)
```

predict

- Under CART, the model uses the honest tree:

```
. crtrees price trunk weight length, seed(12345)
. predict price_hat
```

- Under Random Forest, `crtrees` creates mata matrix file where all trees in the forest are stored (by default this matrix is named `matatrees` and saved in the working directory)

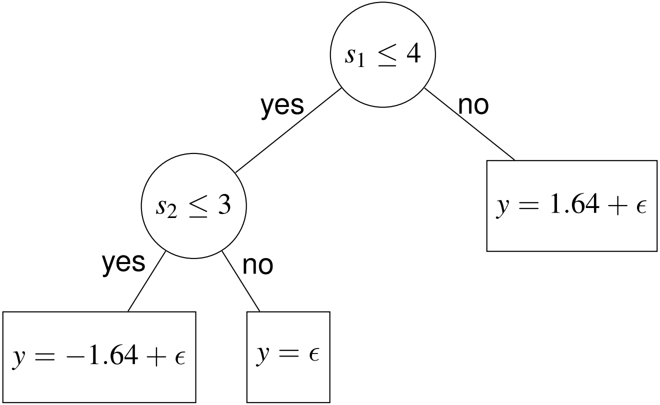
```
. !rm -f mytrees
. crtrees price trunk weight length foreign gear_ratio ///
   in 1/50,reg(weight foreign) stop(5) lssize(0.6) ///
   generate(p_hat) seed(12345) rsplitting(.4) rforests ///
   bootstraps(500) ij savetrees("mytrees")
. predict p_hat2 p_hat_sd in 51/l, opentrees("mytrees")
```

predict (cont'd)

```
Random Forests: Regression
Bootstrap replications (500)
----+--- 100 ---+--- 200 ---+--- 300 ---+--- 400 ---+--- 500
..................................................  500

Dep. Variable = price
Splitting Variables = trunk weight length foreign gear_ratio
Regressors  = weight foreign
Bootstraps  = 500
                                           Number of obs     =         50
                                           R-squared         =     0.7851
                                           Model root SS     =      19466
                                           Residual root SS  =       7141
                                           Total root SS     =      21968

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
       p_hat |         50    6149.417    2780.845   3613.405   13514.61
    p_hat_se |         50    787.3381    680.1503   153.0508   3034.261
Infinitessimal Jacknife Standard Errors

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
      p_hat2 |         24    4114.012    389.0997   3049.175   4691.118
    p_hat_sd |         24    1722.967    1096.124   571.7336    4666.17
```

Simulations

Simulation 1: Regression Trees with constant

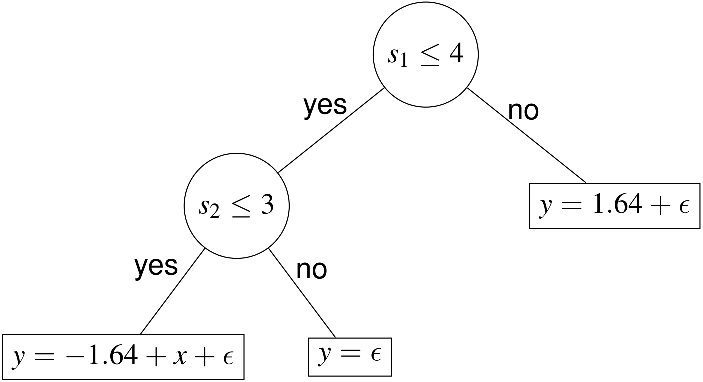


$$\epsilon \sim \mathcal{N}(0, 1)$$
$$s_1 \in \{2, 4, 6, 8\}, s_2 \in \{3, 6, 9, 12\}, s_3 = 0.9 \times s_1$$

crtrees y s1 s2 s3, stop(5) rule(2)

```
Regression Trees with learning and test samples (SE rule: 2)
Learning Sample                                   Test Sample
|T*|          = 3
Number of obs = 524                               Number of obs   =       476
R-squared     = 0.5294                            R-squared       =    0.6102
Avg Dep Var   = 0.637                             Avg Dep Var     =     0.654
Root MSE      = 1.034                             Root MSE        =     0.972
Terminal node results:

Node 3:
     Characteristics:
        6<=s1<=8
     Number of obs =        255
     Average       =   1.638653
     Std.Err.      =     .06302
Node 4:
     Characteristics:
        2<=s1<=4
        s2==3
     Number of obs =         60
     Average       =  -1.600958
     Std.Err.      =      .1316
Node 5:
     Characteristics:
        2<=s1<=4
        6<=s2<=12
     Number of obs =        209
     Average       =   .0571202
     Std.Err.      =     .06808
```

Simulation 2: RT with regression line

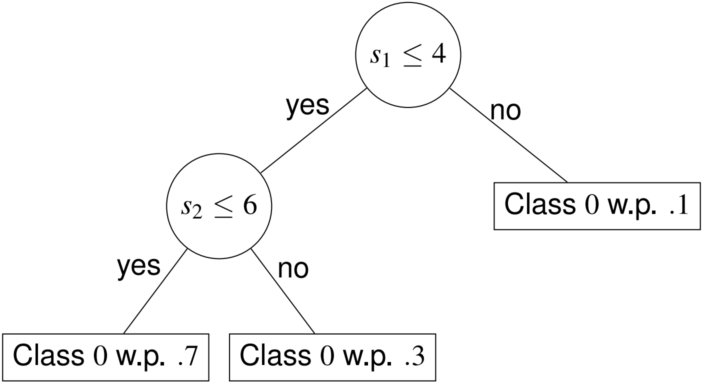


$$x, \epsilon \sim \mathcal{N}(0, 1),$$
$$s_1 \in \{2, 4, 6, 8\}, s_2 \in \{3, 6, 9, 12\}$$

crtrees y s1 s2, reg(x1) stop(5)

```
Regression Trees with learning and test samples (SE rule: 2)
Learning Sample                                   Test Sample
|T*|           = 3
Number of obs  = 504                              Number of obs    =       496
R-squared      = 0.6420                           R-squared        =    0.5200
Avg Dep Var    = 0.620                            Avg Dep Var      =     0.690
Root MSE       = 0.987                            Root MSE         =     1.030
Terminal node results:

Node 3:
     Characteristics:
        6<=s1<=8
     Number of obs =         248
                                                  R-squared        =    0.0121
------------------------------------------------------------------------------
           y |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
           x |    .117363   .0700327     1.68   0.094    -.0198986    .2546246
      _const |   1.758492   .0643814    27.31   0.000     1.632307    1.884677
------------------------------------------------------------------------------

Node 4:
     Characteristics:
        2<=s1<=4
        s2==3
     Number of obs =          76
                                                  R-squared        =    0.5551
------------------------------------------------------------------------------
           y |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
           x |   1.087398   .1084246    10.03   0.000     .8748901    1.299907
      _const |  -1.529997   .1171627   -13.06   0.000    -1.759632   -1.300362
------------------------------------------------------------------------------

Node 5:
     Characteristics:
        2<=s1<=4
        6<=s2<=12
     Number of obs =         180
                                                  R-squared        =    0.0150
------------------------------------------------------------------------------
           y |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
           x |  -.1136537   .0710472    -1.60   0.110    -.2529037    .0255962
      _const |  -.0210631   .0738107    -0.29   0.775    -.1657295    .1236033
------------------------------------------------------------------------------
```

Simulation 3: Classification trees

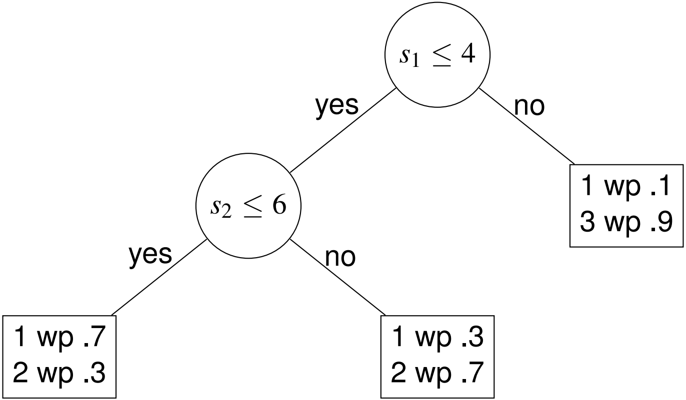


$$\text{Class} \in \{0, 1\}, s_1 \in \{2, 4, 6, 8\}, s_2 \in \{3, 6, 9, 12\}$$

crtrees Class s1 s2, class

```
Classification Trees with learning and test samples (SE rule: 1)
Impurity measure: Gini
Learning Sample                                    Test Sample
Number of obs = 526                                Number of obs   =        474
|T*|          = 3
R(T*)         = 0.1958                             R(T*)           =     0.2229
                                                   SE(R(T*))       =     0.0191
Terminal node results:
Node 3:
     Characteristics:
        6<=s1<=8
     Class predictor =          0
     r(t)            =      0.097
     Number of obs   =        277

Node 4:
     Characteristics:
        2<=s1<=4
        3<=s2<=6
     Class predictor =          1
     r(t)            =      0.289
     Number of obs   =        121

Node 5:
     Characteristics:
        2<=s1<=4
        9<=s2<=12
     Class predictor =          0
     r(t)            =      0.320
     Number of obs   =        128
```

Simulation 4: Classification trees with 3 classes



Class $\in \{1, 2, 3\}$, $s_1 \in \{2, 4, 6, 8\}$, $s_2 \in \{3, 6, 9, 12\}$

crtrees Class s1 s2, class stop(5) rule(0)

```
Classification Trees with learning and test samples (SE rule: 0)
Impurity measure: Gini
Learning Sample                                   Test Sample
Number of obs = 522                               Number of obs  =       478
|T*|          = 3
R(T*)         = 0.1973                            R(T*)           =    0.2038
                                                  SE(R(T*))       =    0.0184
Terminal node results:
Node 3:
     Characteristics:
        6<=s1<=8
     Class predictor =          3
     r(t)            =      0.112
     Number of obs   =        250

Node 4:
     Characteristics:
        2<=s1<=4
        3<=s2<=6
     Class predictor =          1
     r(t)            =      0.311
     Number of obs   =        148

Node 5:
     Characteristics:
        2<=s1<=4
        9<=s2<=12
     Class predictor =          2
     r(t)            =      0.234
     Number of obs   =        124
```

Extensions

- combining splitting variables in a single step
- categorical splitting variables
- graphs producing tree representation and sequences of $R(T)$ estimates
- boosting
- use of random forests for PO-based inference in high-dimensional parameters

Thank you

Biggs, D., B. De Ville, and E. Suen (1991). A method of choosing multiway partitions for classification and decision trees. *Journal of applied statistics 18*(1), 49–62.

Breiman, L. (2001). Random forests. *Machine learning 45*(1), 5–32.

Breiman, L., J. Friedman, R. Olshen, and C. Stone (1984). *Classification and Regression Trees*. Belmont,CA: Wadsworth.

Efron, B. (2014). Estimation and accuracy after model selection. *Journal of the American Statistical Association 109*(507), 991–1007.

Kass, G. V. (1980). An exploratory technique for investigating large quantities of categorical data. *Journal of the Royal Statistical Society: Series C (Applied Statistics) 29*(2), 119–127.

Mentch, L. and G. Hooker (2014). Ensemble trees and clts: Statistical inference for supervised learning. *stat 1050*, 25.

Scornet, E., G. Biau, J.-P. Vert, et al. (2015). Consistency of random forests. *The Annals of Statistics 43*(4), 1716–1741.

Sexton, J. and P. Laake (2009). Standard errors for bagged and random forest estimators. *Computational Statistics & Data Analysis 53*(3), 801–811.