AN IMPLEMENTATION OF CART IN STATA

Ricardo Mora

Universidad Carlos III de Madrid

Madrid, Oct 2015
Outline

1. Introduction
2. Predictive learning
3. CART
4. ARIES
5. Simulations
Introduction
Tree-structured models are predictive models that use two-dimensional binary trees.

- When the target variable can take a finite set of values, binary trees are called classification trees.
- When the target variable can take continuous values (typically real numbers), they are called regression trees.

Estimation of the tree is nontrivial when the structure of the tree is unknown: CART (Breiman et al, 1984)

- CART: **Classification and Regression Trees**

Software packages: Salford Systems CART, Matlab, R

- In Stata, module `<cart>` (Wim van Putten), performs CART analysis for failure time data.

In this presentation, I first describe CART and then discuss its implementation with `<aries>`
Predictive learning
Consider the decomposition of output variable $y$ between the effects of a set of observed controls $x$ and that of all other factors such that

$$y = \mathbb{E}(y|x) + \epsilon$$

The objective in predictive learning is to obtain a useful approximation of $\mathbb{E}(y|x)$

Predictive learning is implemented through an optimization problem on a finite sample $\{y_i, x_i\}_i$ such as

$$\hat{\mathbb{E}}(y|x) = \arg\min_{g(x)} \sum_i (y_i - g(x_i))^2$$
In order to obtain a well defined problem, further assumptions on $g(x_i)$ must be added:
- constraints on eligible functions $g(x_i)$
- constraints on the set of controls $x_i$

Second option not practical in many situations:
- If 100 observations represents a dense sample for a single input system, then for $K$ inputs, $100^K$
- all observations are close to an “edge” of the sample
One way of overcoming these problems is by incorporating a penalty in the problem:

\[
\hat{E} (y|x) = \arg\min_{g(x)} \sum_{i} \left\{ (y_i - g(x_i))^2 + \lambda \phi(g(x_i)) \right\}
\]

The best fit is given by the solution without penalty, \( \lambda = 0 \):

- very low predictive power (overfitting)

Common approach: divide the sample into a learning and a test sample
Examples of predictive learning

- least squares: \( \phi(g(x)) = \begin{cases} \infty & \text{if } g(x) \neq h(x|\theta) \\ 0 & \text{otherwise} \end{cases} \)
  
  - \( h(\cdot) \) and \( \theta \) are known
  
  - hence
  
  \[ \hat{E}(y|x) = \arg\min_{g(x)} \sum_i \left\{ (y_i - h(x_i|\theta))^2 \right\} \]

- single layer neural network: \( g(x) = \sum_t a_t s(x'\theta_t) \)
  
  - \( s(\cdot) \) is a sigmoid function

- projection pursuit: \( g(x) = \sum_t g_t(x'\theta_t|a_t) \)
Tree structures

$$\phi (g (x)) = \begin{cases} \infty \text{ if } g (x) \neq \sum_{t \in T} a_t \times \prod_{j=1}^{K} \mathbb{1} (l_j < x_j \leq u_j) \\ 0 \text{ otherwise} \end{cases}$$

where $l_j$ and $u_j$ are the respective lower and upper limit of the region on each control

- $T$ is a partition of the space of all possible values of $x$
- Therefore

$$E (y|x) = a_t \times \prod_{j=1}^{K} \mathbb{1} (l_j < x_j \leq u_j)$$

- Both the partition $T$ and the expectations $a_t$ associated to each element in the partition are unknown
Example
Mathematical and tree representation

\[ E(y|x_1, x_2) = \begin{cases} 
  a_1 & \text{if } x_2 \leq x_{21} \\
  a_2 & \text{if } x_2 > x_{21} \text{ and } x_1 \leq x_{11} \\
  a_3 & \text{if } x_2 > x_{21} \text{ and } x_1 > x_{11} 
\end{cases} \]
Classification And Regression Trees
The problem is if we know the tree structure: least squares

Least squares is unfeasible when structure is unknown

- LS on 50 cells with at most two terminal nodes $\approx 6 \times 10^{14}$ models (or more than 15 years of computing time)

Second best solution: recursive partition

- regions become more local
- each step only considers a limited number of possible splits
Assume that we have a tree structure $T$ and that we want to split node $t^*$, one terminal node in $T$.

Let $R(T)$ be the residual sum of squares within each terminal node of the tree.

Consider the set of possible binary partitions or splits.

Recursive partitioning is defined by choosing the split at each step of the algorithm such that the reduction in $R(T)$ is maximized.

The process 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).
Growing the tree until $T_{MAX}$

- Often, the result will be equivalent to dividing the sample into all possible cells and computing within-cell least squares.
- Growing the tree until no further partitioning is possible helps avoiding having to select a rule to stop splitting.
- Usually, however, $T_{MAX}$ will be too complex in the sense that some terminal nodes could be aggregated into one terminal node.
- 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.
- It is also intuitive to see that if aggregation goes too far, aggregation bias will become a serious problem.
Pruning the tree: Error-complexity clustering

- In order to aggregate from $T_{MAX}$ we can use a clustering algorithm procedure.
- For a given value $\alpha$, let $R(\alpha, T) = R(T) + \alpha |T|$ where $|T|$ denotes the number of terminal nodes, or complexity, of the tree.
- The tree structured estimate for a given $\alpha$, $T(\alpha)$, is the value that minimizes $R(\alpha, T)$ for the set of subtrees of $T_{MAX}$.
  - $T(\alpha)$ belongs to a much broader set than the sequence of all trees obtained in the recursive partition algorithm.
- For all $\alpha$: $T_{MAX} \succ T(\alpha_1) \succ \ldots \succ \{\text{root}\}$ (pruning the tree)
Honest tree

- By construction, $R(T_{MAX})$ is the lowest value for the sequence of subtrees.
  - This may not be true for an independent sample: choosing $T_{MAX}$ as our tree structured model may lead to overoptimistic results for $R(\cdot)$
- There are three strategies to obtain unbiased estimates of $R(\cdot)$:
  - test sample: choose the tree in the sequence that minimizes
    $$R^{ts}(T) + s \times SE(R^{ts}(T))$$
    where $s$ is a given positive value
  - $K$-fold cross validation
  - bootstrap
$T_{\text{MAX}}$ example: 5 terminal nodes
$T_1$ example: 4 terminal nodes
$T_2$ example: 1 terminal node

- The sequence is thus: $\{T_{MAX}, T_1, T_2 \equiv \{\text{root}\}\}$
- Among the three, we would choose the tree that gives a smaller $R^{ts}(T) + s \times SE(R^{ts}(T))$
  - For example, $s = 1$ may be useful when the sequence provides a flat profile for $R^{ts}(T)$ after reaching a certain level of complexity
CART Estimator properties

- Consistency requires an ever more dense sample at all \( n \)-dimensional balls of the input space.
- Cost-complexity minimization together with test sample unbiased estimates of \( R(\cdot) \) guarantee that such condition is satisfied by regression tree partitions.
- The basic results can be found in Breiman et alia (1984, chapter 12).
- For small samples, high correlation in the explanatory variables will induce instability in the tree topology: interpretation of the contribution of each variable will become problematic.
The aries ado

```
aries varname splitvarlist [if] [in], options
```

- `varname`: output variable (it must be discrete if classification tree is performed)
- `splitvarlist`: variables whose combinations identify the terminal nodes

By default, the command performs CART for regression trees with a constant in each terminal node using a test sample and the 0 SE rule for estimating the honest tree.
Options for regression trees

- **regressors**(varlist): controls in terminal nodes. A regression line is estimated in each terminal node.
- **exogenous**(varlist): list of exogenous variables. IV regression is estimated in each terminal node. The number of exogenous variables must be at least equal to the number of controls.
- **noconstant**: estimates regression lines without constant.
Options for classification trees

Classification trees:
- The output variable must be discrete.
- Each value of the output variable refers to one of $J$ classes.
- Classification trees grows the tree using a given impurity measure based on the sample probability of each class in each node.

Options for classification trees:
- **classification**: performs classification tree (output variable must be discrete)
- **impurity(#)**: impurity measure code:
  - 1: Entropy measure
  - 2: Gini measure
Options common to classification and regression trees

- **seed(#)**: seed to replicate random division of the sample into a learning and a test sample
- **lssize(#)**: proportion of the learning sample (default is 0.5)
- **stop(#)**: integer for stop splitting rule
- **rule(#)**: SE rule to identify honest tree
Output display

After regression trees:
- The overall fit of the model both for the learning and the test sample
- The definition of each terminal node in terms of the splitting variables
- The coefficient estimates and standard errors for each terminal node
  - The standard error of each terminal node regression is computed using the test sample

After classification trees:
- The overall miss-classification rate of the model estimated by test sample
- The definition of each terminal node in terms of the splitting variables
- The miss-classification rate for each terminal node in the learning and the test sample
Saved results

- Saved results for Regression trees:
  - usual scalars saved in e() after regression
  - coefficients estimates and variance-covariance matrices for each terminal node’s regression

- Common saved results:
  - a matrix representation of the tree structure
  - a matrix with range of values for splitting variables in each terminal node
  - a matrix with the sequence of optimal trees and the test-sample $R_{ts}^2(T)$ measure for each of them
**Predictions**

- *aries* saves the coefficients estimates and also matrix representations of the estimated tree.
- *predict* is available after estimation.

**After regression trees:**

- *predict newvar [if] [in], xb*
- residual nodes
  - *xb*: output variable predictions (the default)
  - *residual*: residuals
  - *nodes*: terminal node code

**After classification trees:**

- *predict newvar [if] [in]*
  - the variable *newvar* includes the class code predicted by the estimated tree for each observation.
Simulations
Example 1: RT with constant 

$y = -3 + \epsilon$

$y = \epsilon$

$\epsilon \sim \mathcal{N}(0, 1)$

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

aries y s1 s2, stop(5)

Learning Sample

Number of obs = 522
F( 2, 519) = 1135.5
Prob > F = 0.0000
R-squared = 0.8140
Adj R-squared = 0.8133
Root MSE = 1.0217

Node 3: 6<=s1<=8 3<=s2<=12
No of obs (Learning smpl) = 257
No of obs (Test smpl) = 239

| Coef.  | Std. Err. | z     | P>|z|  | [95% Conf. Interval] |
|--------|-----------|-------|------|---------------------|
| _cons  | 3.109557  | 0.0622225 | 49.97 | 0.000 | 2.987603 - 3.23151 |

Node 4: 2<=s1<=4 3<=s2<=3
No of obs (Learning smpl) = 70
No of obs (Test smpl) = 63

| Coef.  | Std. Err. | z     | P>|z|  | [95% Conf. Interval] |
|--------|-----------|-------|------|---------------------|
| _cons  | -2.852275 | 0.1054995 | -27.04 | 0.000 | -3.059051 - 2.6455 |

Node 5: 2<=s1<=4 6<=s2<=12
No of obs (Learning smpl) = 195
No of obs (Test smpl) = 176

| Coef.  | Std. Err. | z     | P>|z|  | [95% Conf. Interval] |
|--------|-----------|-------|------|---------------------|
| _cons  | -0.0097753 | 0.0760195 | -0.13 | 0.898 | -0.1587707 - 0.1392202 |

Test Sample

Number of obs = 478
F( 2, 475) = 1066.1
Prob > F = 0.0000
R-squared = 0.8178
Adj R-squared = 0.8170
Root MSE = 1.0077

Node 3: 6<=s1<=8 3<=s2<=12
No of obs (Learning smpl) = 257
No of obs (Test smpl) = 239

| Coef.  | Std. Err. | z     | P>|z|  | [95% Conf. Interval] |
|--------|-----------|-------|------|---------------------|
| _cons  | 3.109557  | 0.0622225 | 49.97 | 0.000 | 2.987603 - 3.23151 |

Node 4: 2<=s1<=4 3<=s2<=3
No of obs (Learning smpl) = 70
No of obs (Test smpl) = 63

| Coef.  | Std. Err. | z     | P>|z|  | [95% Conf. Interval] |
|--------|-----------|-------|------|---------------------|
| _cons  | -2.852275 | 0.1054995 | -27.04 | 0.000 | -3.059051 - 2.6455 |

Node 5: 2<=s1<=4 6<=s2<=12
No of obs (Learning smpl) = 195
No of obs (Test smpl) = 176

| Coef.  | Std. Err. | z     | P>|z|  | [95% Conf. Interval] |
|--------|-----------|-------|------|---------------------|
| _cons  | -0.0097753 | 0.0760195 | -0.13 | 0.898 | -0.1587707 - 0.1392202 |
## A simple Monte Carlo

### Table: Monte Carlo: $R^2$

<table>
<thead>
<tr>
<th>No. obs.</th>
<th>$\sigma$</th>
<th>OLS</th>
<th>aries:LS</th>
<th>aries:TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td>.5</td>
<td>0.711</td>
<td>0.946</td>
<td>0.946</td>
</tr>
<tr>
<td>250</td>
<td>1</td>
<td>0.612</td>
<td>0.814</td>
<td>0.815</td>
</tr>
<tr>
<td>250</td>
<td>2</td>
<td>0.396</td>
<td>0.525</td>
<td>0.528</td>
</tr>
<tr>
<td>750</td>
<td>.5</td>
<td>0.710</td>
<td>0.946</td>
<td>0.947</td>
</tr>
<tr>
<td>750</td>
<td>1</td>
<td>0.611</td>
<td>0.813</td>
<td>0.816</td>
</tr>
<tr>
<td>750</td>
<td>2</td>
<td>0.393</td>
<td>0.520</td>
<td>0.529</td>
</tr>
<tr>
<td>1000</td>
<td>.5</td>
<td>0.711</td>
<td>0.946</td>
<td>0.946</td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td>0.612</td>
<td>0.814</td>
<td>0.815</td>
</tr>
<tr>
<td>1000</td>
<td>2</td>
<td>0.393</td>
<td>0.523</td>
<td>0.524</td>
</tr>
</tbody>
</table>

Note: Monte Carlo results using 500 replications.
Example 2: RT with regression line

\[ y = -3 + 0.5 \times x_1 + \epsilon \]

\[ s_1 \leq 4 \]

\[ s_2 \leq 3 \]

\[ \epsilon \sim \mathcal{N}(0, 1) \]

\[ s_1 \in \{2, 4, 6, 8\}, s_2 \in \{3, 6, 9, 12\} \]
Simulations

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

Learning Sample
Number of obs = 522
F(  5,  516) = 339.95
Prob > F = 0.0000
R-squared = 0.7671
Adj R-squared = 0.7649
Root MSE = 1.0113

Node 3:  6<=s1<=8  3<=s2<=12
No of obs (Learning smpl) = 257
No of obs (Test smpl) = 239

| Coef.    | Std. Err. | z    | P>|z|   | [95% Conf. Interval] |
|----------|-----------|------|-------|---------------------|
| x1       | -.0738962 | .0543166 | -1.36 | 0.174               | -.1803548  .0325624 |
| _cons    | 3.229408  | .1481916 | 21.79 | 0.000               | 2.938958   3.519859 |

Node 4:  2<=s1<=4  3<=s2<=3
No of obs (Learning smpl) = 70
No of obs (Test smpl) = 63

| Coef.    | Std. Err. | z    | P>|z|   | [95% Conf. Interval] |
|----------|-----------|------|-------|---------------------|
| x1       | .5736499  | .1053622 | 5.44 | 0.000               | .3671437   .780156 |
| _cons    | -3.173849 | .2846731 | -11.15| 0.000              | -3.731798  -2.6159 |

Node 5:  2<=s1<=4  6<=s2<=12
No of obs (Learning smpl) = 195
No of obs (Test smpl) = 176

| Coef.    | Std. Err. | z    | P>|z|   | [95% Conf. Interval] |
|----------|-----------|------|-------|---------------------|
| x1       | .0250965  | .0669443 | 0.37 | 0.708               | -.106112   .1563049 |
| _cons    | .054355   | .1829895 | 0.30 | 0.766               | -.3042978  .4130078 |
Some saved results

```plaintext
matrix list e(tree)

<table>
<thead>
<tr>
<th>Node</th>
<th>Child</th>
<th>Split_var</th>
<th>Cut_off</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>r2</td>
<td>2</td>
<td>2</td>
<td>4.5</td>
</tr>
<tr>
<td>r3</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>r4</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>r5</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

matrix list e(_tree)

<table>
<thead>
<tr>
<th>Node</th>
<th>s1_min</th>
<th>s1_max</th>
<th>s2_min</th>
<th>s2_max</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>r2</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>r3</td>
<td>3</td>
<td>6</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>r4</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>r5</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

matrix list e(pruning)

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Impurity</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>4.2953238</td>
</tr>
<tr>
<td>r2</td>
<td>1.3388132</td>
</tr>
<tr>
<td>r3</td>
<td>0.94715324</td>
</tr>
<tr>
<td>r4</td>
<td>0.94849036</td>
</tr>
<tr>
<td>r5</td>
<td>0.95739838</td>
</tr>
<tr>
<td>r6</td>
<td>0.98367937</td>
</tr>
<tr>
<td>r7</td>
<td>0.99016994</td>
</tr>
<tr>
<td>r8</td>
<td>0.98654955</td>
</tr>
<tr>
<td>r9</td>
<td>0.9930246</td>
</tr>
<tr>
<td>r10</td>
<td>0.99768252</td>
</tr>
<tr>
<td>r11</td>
<td>0.99777621</td>
</tr>
<tr>
<td>r12</td>
<td>0.99712844</td>
</tr>
</tbody>
</table>
```
Example 3: RT with IV line

\[ s_1 \leq 4 \]

\[ s_2 \leq 3 \]

\[ y = -3 + 0.5 \times x_1 + \epsilon \]

\[ y = \epsilon \]

\[ \epsilon \sim \mathcal{N}(0, 1), \text{cov}(x_1, \epsilon) \neq 0, \text{cov}(z_1, \epsilon) = 0 \]

\[ s_1 \in \{2, 4, 6, 8\}, s_2 \in \{3, 6, 9, 12\} \]
```plaintext
aries y s1 s2, reg(x1) exog(z1) stop(5)

Learning Sample
Number of obs = 522
F(  5,  516) = 386.75
Prob > F = 0.0000
R-squared = 0.7899
Adj R-squared = 0.7879
Root MSE = 1.0190

No of obs (Learning smpl) = 257
Coef. Std. Err. z P>|z| [95% Conf. Interval]
---
x1   -.152294  .1119791 -1.36 0.174  -.371769   .0671811
_cons 3.236396  .1529517 21.16 0.000   2.936616   3.536176

Exogenous variable: z1

Node 3:  6<=s1<=8 3<=s2<=12
No of obs (Test smpl) = 239

Coef. Std. Err. z P>|z| [95% Conf. Interval]
---
x1   .6430845  .2088366  3.08 0.002    .2337722   1.052397
_cons -3.168874  .2838073 -11.17 0.000   -3.725126   -2.612622

Exogenous variable: z1

Node 4:  2<=s1<=4 3<=s2<=3
No of obs (Learning smpl) = 70
No of obs (Test smpl) = 63

Coef. Std. Err. z P>|z| [95% Conf. Interval]
---
x1   .0496941  .1334359  0.37 0.710   -.2118355   .3112237
_cons .0538148  .1855386  0.29 0.772   -.3098342   .4174638

Exogenous variable: z1

Node 5:  2<=s1<=4 6<=s2<=12
No of obs (Learning smpl) = 195
No of obs (Test smpl) = 176

Coef. Std. Err. z P>|z| [95% Conf. Interval]
---
x1   .0496941  .1334359  0.37 0.710   -.2118355   .3112237
```

Exogenous variable: z1
Example 4: Classification trees

\[
s_1 \leq 4
\]

\[
s_2 \leq 3
\]

\[
s_1 \in \{2, 4, 6, 8\}, \ s_2 \in \{3, 6, 9, 12\}
\]
<table>
<thead>
<tr>
<th>Node</th>
<th>Condition</th>
<th>Class</th>
<th>Learning Sample</th>
<th>Test Sample</th>
<th>Pr(missclassification)</th>
<th>No. of obs.</th>
<th>Learning Sample</th>
<th>Test Sample</th>
<th>Pr(missclassification)</th>
<th>No. of obs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>$6 \leq s_1 \leq 8$  $3 \leq s_2 \leq 12$</td>
<td>3</td>
<td>0.2918</td>
<td>0.2762</td>
<td></td>
<td>257</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$2 \leq s_1 \leq 4$  $3 \leq s_2 \leq 3$</td>
<td>1</td>
<td>0.3000</td>
<td>0.2857</td>
<td></td>
<td>63</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>$2 \leq s_1 \leq 4$  $12 \leq s_2 \leq 12$</td>
<td>2</td>
<td>0.2373</td>
<td>0.3729</td>
<td></td>
<td>59</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>$2 \leq s_1 \leq 2$  $6 \leq s_2 \leq 9$</td>
<td>2</td>
<td>0.2329</td>
<td>0.3770</td>
<td></td>
<td>61</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>$4 \leq s_1 \leq 4$  $6 \leq s_2 \leq 9$</td>
<td>2</td>
<td>0.3333</td>
<td>0.3393</td>
<td></td>
<td>56</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Simulations
aries y s1 s2, class
Simulations

\[ s_1 \leq 4 \]

\[ s_2 \leq 3 \]

Class 1

Yes

No

Class 2

Class 2

5

Yes

No

Class 3

Class 2

10
Extensions

- $\nu$-fold cross-validation for small data sets
- combining splitting variables in a single step
- categorical splitting variables
- graphs producing tree representation and sequence of $R^{ts}(T)$ estimates
- alternative impurity measurements
- boosting
Thank you