Classification And Regression Trees: A Practical Guide for Describing a Dataset

Leo Pekelis

February 2nd, 2013, Bicoastal Datafest, Stanford University
What is a Tree?
What is a Tree?
What is a (binary) Decision Tree?
What is a (binary) Decision Tree? Example

Classifying SPAC Donation Size

- The data is all donations to SPACs in excess of $200, by early 2012, from fec.gov
The Structural Model

- $F(x) = \sum_{i=1}^{M} c_m I(x \in R_m)$

- $\{R_m\}_{1}^{M}$ are subregions of the input variable space, and $x$ is a vector of input variables.
  - Examples: $\{x_9 < 15.2\}$, $\{9 \leq x_{300} < 786 \& \text{color} = \text{red}\}$

- $c_m$ are the estimated values of the outcome ($y$) in region $R_m$

- CART tries to minimize
  - $e(T) = \sum_{i=1}^{N} \left[ y_i - \sum_{m=1}^{M} c_m I(x \in R_m) \right]^2$
  - with respect to $c_m$ and $R_m$
Some Important Facts about CART

1. The $R_m$ regions are disjoint and rectangular
   - giving a piecewise constant approximation to the true $F(x)$

2. CART doesn’t find the “best” regions exactly
   - uses recursive partitioning, or a greedy stepwise descent

3. Both simplifications are to simplify a combinatorially hard problem and make it solvable in reasonable time.
   - also allows for natural representations of regions as a binary decision tree
How do we run it?

```r
# install the package to R
install.packages("rpart", repos = "http://cran.us.r-project.org")
#
## The downloaded binary packages are in
## /varfolders/0m/xzr0fktj78sgl36y77z34djr0000gn/T//RtmpPUWHm/downloaded_packages

# load the library
library(rpart)

# load the dataset
load("spac.Rdata")

spac.tree = rpart(Donation ~ ., data = spac.data, cp = 10^(-6))

### the function arguments:
### 1) formula, of the form: outcome ~ predictors
# note: outcome ~ . is 'use all other variables in data'
### 2) data: a data.frame object, or any matrix which has variables as
# columns and observations as rows
### 3) cp: used to choose depth of the tree, we'll manually prune the tree
# later and hence set the threshold very low (more on this later)

# The commands, print() and summary() will be useful to look at the tree.
# But first, lets see how big the created tree was

# The object spac.tree is a list with a number of entries that can be
# accessed via the $ symbol. A list is like a hash table.

# To see the entries in a list, use names()
names(spac.tree)

## [1] "frame" "where" "call"
## [4] "terms" "cptable" "method"
## [7] "parms" "control" "functions"
## [10] "numresp" "splits" "csplit"
## [13] "variable.importance" "y" "ordered"

# Within spac.tree the ctable will tell us a little about the size of the
# tree
spac.tree$cptable[1:10, ]

##   CP nsplit rel error xerror xstd
## 1 0.037317    1 1.0000 1.0000 0.3477
## 2 0.016462    2 0.9254 1.0784 0.3493
## 3 0.003617    6 0.8595 1.0683 0.3300
## 4 0.002751    8 0.8523 1.0510 0.3171
## 5 0.001581    9 0.8495 1.0500 0.3170
## 6 0.001516   17 0.8369 1.0643 0.3170
## 7 0.001470   21 0.8305 1.0643 0.3170
## 8 0.001454   27 0.8217 1.0663 0.3170
## 9 0.001432   29 0.8188 1.0663 0.3170
##10 0.001020   32 0.8145 1.0693 0.3170

# ...
spac.tree$cptable[dim(spac.tree$cptable)[1] - 9:0, ]

##   CP nsplit rel error xerror xstd
## 84 1.901e-06   169 0.7951 1.0670 0.3133
```
that's a lot of splits! I'm going to prune the tree to 9 splits

```r
cp9 = which(spac.tree$cp.table[, 2] == 9)
spac.tree9 = prune(spac.tree, spac.tree$cp.table[cp9, 1])
```

# now lets look at the tree with print() and summary()

```r
print(spac.tree9)
```

```r
Call:  
 rpart(formula = Donation ~ ., data = spac.data, cp = 10^-6)  

## n= 3668  
## node), split, n, deviance, yval  
## * denotes terminal node  
## 1) root 3668 1.438e+14 30940.0  
## 2) NV=0 3627 9.400e+12 28140.0  
## 4) FID=otherFID,C00487470,C00488403,C00499335 3239 8.088e+13 20110.0  
## 8) smbiz=0 3027 2.897e+13 16380.0  
## 16) blank=0 2467 1.580e+13 10930.0 *  
## 17) blank=1 560 1.278e+13 40370.0 *  
## 9) smbiz=1 212 5.126e+13 73400.0  
## 18) TX=0 165 1.867e+12 26050.0 *  
## 19) TX=1 47 4.772e+13 239600.0  
## 38) FID=C00488403,C00499335 31 5.142e+06 567.2 *  
## 39) FID=otherFID 16 4.252e+13 702800.0 *  
## 5) FID=C00490045 388 1.117e+13 95130.0  
## 10) NY=0 345 6.533e+12 82900.0 *  
## 11) NY=1 43 4.176e+12 193300.0  
## 22) Day< 27.5 35 2.033e+12 138000.0 *  
## 23) Day>=27.5 8 1.568e+12 435000.0 *  
## 3) NV=1 41 4.723e+13 278400.0  
## 6) Month>=3 32 3.476e+11 41390.0 *  
## 7) Month< 3 9 3.869e+13 1121000.0 *
```

```r
summary(spac.tree9)
```

```r
## Call:  
## rpart(formula = Donation ~ ., data = spac.data, cp = 10^-6)  
## n= 3668  
##   CP nsplit rel error xerror xstd  
## 1 0.037317    0  1.0000  1.000 0.3477  
## 2 0.016462    2  0.9254  1.078 0.3493  
## 3 0.003617    6  0.8595  1.068 0.3300  
## 4 0.002751    8  0.8523  1.051 0.3171  
## 5 0.001581    9  0.8495  1.050 0.3170  
##   Variable importance  
## Month  FID  NV  TX  tech  oil  doctor  writing  smbiz  
## 35  28  9  6  3  3  3  3  3  2  
##   Day  NY  blank  biz  
##  2  2  1  1  1  
## 1) root 3668 observations,  complexity param=0.03732  
##   mean=3.094e+04, MSE=3.919e+10  
## left son=2 (3627 obs) right son=3 (41 obs)  
## Primary splits:  
## NV   splits as LR, improve=0.017660, (0 missing)  
## FID splits as LRLLL, improve=0.012390, (0 missing)  
## Month < 5.5 to the right, improve=0.005567, (0 missing)
## left son=18 (165 obs) right son=19 (47 obs)
## Primary splits:
## TX splits as LR, improve=0.032550, (0 missing)
## Month < 1.5 to the right, improve=0.017010, (0 missing)
## FID splits as R-LLL, improve=0.009249, (0 missing)
## Day < 28.5 to the left, improve=0.007682, (0 missing)
## professional splits as RL, improve=0.002284, (0 missing)
## Surrogate splits:
## FID splits as L-LRL, agree=0.892, adj=0.511, (0 split)
## teach splits as LR, agree=0.783, adj=0.021, (0 split)
## oil splits as LR, agree=0.783, adj=0.021, (0 split)
##
## Node number 10: 345 observations
## mean=8.29e+04, MSE=1.894e+10
##
## Node number 11: 43 observations, complexity param=0.003617
## mean=1.933e+05, MSE=9.711e+10
##
## Node number 16: 2467 observations
## mean=1.093e+04, MSE=6.405e+09
##
## Node number 17: 560 observations
## mean=4.037e+04, MSE=2.282e+10
##
## Node number 18: 165 observations
## mean=2.605e+04, MSE=1.131e+10
##
## Node number 19: 47 observations, complexity param=0.01646
## mean=2.396e+05, MSE=1.015e+12
##
## Node number 22: 35 observations
## mean=1.38e+05, MSE=5.809e+10
##
## Node number 23: 8 observations
## mean=4.35e+05, MSE=1.961e+11
##
## Node number 38: 31 observations
## mean=567.2, MSE=1.659e+05
##
## Node number 39: 16 observations
## mean=7.028e+05, MSE=2.658e+12
##
## finally, let's get a graphical representation of the tree, and save to a
## png file
##
## post("spactree9.png", width = 1200, height = 800)
##
## dev.off()
How do we run it? The graphical representation.
What about exporting the results?

```r
# will use a combination of list entries: frame, splits, and csplit
spac.tree9$frame[1:5,]
```

<table>
<thead>
<tr>
<th>Var</th>
<th>N</th>
<th>WT</th>
<th>Dev</th>
<th>Yval</th>
<th>Complexity</th>
<th>Ncomplete</th>
<th>Nsurrogate</th>
</tr>
</thead>
<tbody>
<tr>
<td>NV</td>
<td>3668</td>
<td>1.438e+14</td>
<td>30936</td>
<td>0.037317</td>
<td>4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>FID</td>
<td>3627</td>
<td>9.400e+13</td>
<td>28138</td>
<td>0.016462</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>smbiz</td>
<td>3239</td>
<td>8.088e+13</td>
<td>20113</td>
<td>0.016462</td>
<td>4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>blank</td>
<td>3027</td>
<td>2.897e+13</td>
<td>16381</td>
<td>0.002751</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>&lt;leaf&gt;</td>
<td>2467</td>
<td>1.580e+13</td>
<td>10935</td>
<td>0.001581</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

```r
# frame is a matrix with 1 row per node of the tree
# row name corresponds to a unique node index
# var - name of the variable used in the split, or <leaf>
# n - number of observations reaching the node
# yval - the fitted outcome value at the node

```r
spac.tree9$splits[1:5,]
```

<table>
<thead>
<tr>
<th>Var</th>
<th>Count</th>
<th>Ncat</th>
<th>Improve</th>
<th>Index</th>
<th>Adj</th>
</tr>
</thead>
<tbody>
<tr>
<td>NV</td>
<td>3668</td>
<td>2</td>
<td>0.017664</td>
<td>1.0</td>
<td>0</td>
</tr>
<tr>
<td>FID</td>
<td>3668</td>
<td>5</td>
<td>0.012395</td>
<td>2.0</td>
<td>0</td>
</tr>
<tr>
<td>Month</td>
<td>3668</td>
<td>1</td>
<td>0.005567</td>
<td>5.5</td>
<td>0</td>
</tr>
<tr>
<td>smbiz</td>
<td>3668</td>
<td>2</td>
<td>0.004716</td>
<td>3.0</td>
<td>0</td>
</tr>
<tr>
<td>retired</td>
<td>3668</td>
<td>2</td>
<td>0.003653</td>
<td>4.0</td>
<td>0</td>
</tr>
</tbody>
</table>

# splits characterizes the splits making the regions Rm
# row name is the variable being split
# count - the number of observations coming into the split
# ncat - number of categories of categorical variable, or 1 if the 
# variable is numeric
# improve - the improvement in the objective using the split
# index - either the row number of the csplit matrix (for categorical 
# variables), or the value of the optimal split (for numeric variables)

```r
spac.tree9$csplit[1:5,]
```

```r
[1,] 1 3 2 2 2
[2,] 1 3 1 1 1
[3,] 1 3 2 2 2
[4,] 3 1 2 2 2
[5,] 1 3 1 1 1
```

# has 1 row for each split on a categorical variable
# the row number corresponds to index in spac.tree11$splits above
# each column is an ordered level of a categorical variable, up to the max
# levels of any categorical var
# an entry of 1 - that level goes left in the split
# 3 - that level goes right in the split
# 2 - that level is not included in the split
What about exporting the results?

- To recreate a decision tree, you would at least extract the following columns of information:
  
  - `rownames(spac.tree9$splits)`
  
  - `spac.tree9$splits[, "count"], spac.tree9$splits[, "index"] and spac.tree9$splits[, "ncat"]`
  
  - `spac.tree9$frame[, "var"], spac.tree9[, "n"] and spac.tree9$frame[, "yval"]`
  
  - `spac.tree9$csplit corresponding to the rows given by "index" where "ncat" > 2 in "splits"`
  
- The order of splits in "frame" are depth first, and left branch first

- Match between "frame" and "splits" by variable name and number of observations
  
  - since a variable can be split multiple times, and frame also includes competing and surrogate splits
Automatic Way to Select Tree Size

- Can calculate contribution of split to decreasing objective $e(T)$ by
  
  \[ e_m = \frac{1}{N} \sum_{x_i \in R_m} (y_i - \bar{y}_m)^2 \]

- $Imp_m = e_m - e_{ml} - e_{mr}$

- If $Imp_m \geq cp$ then accept the split, otherwise make $m$ a terminal node
  
  - $cp > 0$ is a tuning parameter, giving tree sizes as in “cptable”
  
  - Actually a little trickier because the rule is applied in inverse order of depth

- Solves the problem:

\[
\min_T [e(T) + cp|T|]
\]

- where $|T|$ is the number of terminal nodes of the tree
Automatic Way to Select Tree Size

- The entry "cptable" gives tree statistics for each \( cp \)

- "rel error" is the ratio of the objective, \( e(T) \), to that of a single root tree
  
  - This is always decreasing with \( cp \)

- "xerror" is the average of 10 fold cross validation error
  
  - i.e. leave out 1/10th of the dataset,
    
    - train a size n tree on the other 9/10ths,
    - and compute \( e(T) \) on the left out part
  
  - this is more useful for prediction, and not as useful to us for describing a dataset

- Can be thought of as a measure of **pervasiveness**

- Could consider a criteria that penalizes large trees
  
  - Not unreasonable: \( N \times (relerror) + 2|T| \)
Automatic Way to Select Tree Size

```r
which.min(spac.tree$cp$table[, 4])
## 1
## 1
```

# gives a value of 1, meaning none of the splits are 'pervasive'

# but using the criteria above, penalizing large trees
```r
cpstat = dim(spac.data)[1] * spac.tree$cp$table[, 3] + 2 * (spac.tree$cp$table[, 2] + 1)
round(spac.tree$cp$table[which.min(cpstat), ], 3)
```

```r
## CP  nsplit  rel error  xerror  xstd
## 0.001 39.000 0.808 1.064 0.313
```

# suggests a tree size with 39 splits
Advantages of Trees

1. **Fast** computations

2. **Invariant** under monotone transformations of variables
   - *Scaling doesn’t matter!*
   - *Immune to outliers in x*

3. **Resistance** to irrelevant variables, so can throw lots of variables into it

4. **One tuning parameter** (tree size, or cp)

5. **Interpretable** model representation

6. **Handles missing data** by keeping track of surrogate, or highly correlated, backup splits at every node

7. Extends to **categorical outcomes** easily
Disadvantages of Trees

1. **Accuracy**
   - $F(x)$ may not be piecewise constant (but decent overall approximation)
   - Data Fragmentation (ok, if you have lots of data)
   - $F(x)$ must involve high order interactions

2. **Variance**
   - Each subsequent split depends on the previous ones, so an error in a higher split is propagated down.
   - Small change in dataset can cause big change in tree
     - If you only have a random sample of a population, this can be a problem.
     - Not as much of an issue if you’re describing a dataset
CART libraries outside of R: weka

- weka 3: Data mining software in JAVA
- Relevant class `weka.classifiers.trees.J48`
- Simple command line syntax

  ```
  java weka.classifiers.trees.J48 -t data/weather.arff -i
  ```

- ARFF is *Attribute-Relation File Format* and data format for weka
- `weka.core.converters` package contains converters for usual data files
- Also call classes directly

```java
import weka.core.Instances;
import weka.classifiers.Evaluation;
import weka.classifiers.trees.J48;
...
Instances train = ... // from somewhere
Instances test = ... // from somewhere
// train classifier
Classifier cls = new J48();
cls.buildClassifier(train);
// evaluate classifier and print some statistics
Evaluation eval = new Evaluation(train);
eval.evaluateModel(cls, test);
System.out.println(eval.toSummaryString("", false));
```

- `weka.gui.treevisualizer.TreeVisualizer` class to visualize trees
CART libraries outside of R: orange

- orange: Data mining through visual programming or Python scripting.
- http://orange.biolab.si/
- has proprietary tab-delimited data format
  - Can import from csv, but is not very robust
  - More info: /Orange.dataformats/
- Relevant function: Orange.regression.tree.TreeLearner(...)
- Visualizing trees: Orange renders trees in dot - plain text graph description language readable by both human and computer
  - tree.dot(file_name="0.dot", node_shape="ellipse", leaf_shape="box")
CART libraries outside of R: opencv

- opencv: (Open Source Computer Vision) is a library of programming functions for real time computer vision, in C++
  - http://opencv.willowgarage.com/wiki/
- Uses n-dimensional array class Mat to store and operate on data
  - core_basic_structures.html#mat
- CvDTree class is an honest representation of CART algorithm
  - ml_decision_trees.html
  - mushroom.cpp example file demonstrates how to use decision trees
References


- Jerome Friedman’s 315b course notes
Two solutions to Disadvantages (extra slides)

1. **Boosted Trees, aka Forests, MART**
   - \( F(x) = \sum_{k=1}^{K} a_k f(x; c_m^k, R_m^k) \)
   - *Now each \( f() \) is a tree, and \( F() \) is a linear combination of trees*
   - *Each tree can model an additive effect, or many low order interactions*
   - *Variance of a combination of identically distributed objects is lower than any individual*
   - *Disadvantage: loses decision tree interpretability unless \( K \) is small*

2. **Random Forests**
   - *Similar to boosted trees, but now random subsets of the data are used for each tree*
   - *Simpler to fit than boosted trees*
   - *Accuracy is usually somewhere in between a single tree and boosted trees*
How are Boosted Trees Interpreted? (extra slides)

- Relative Importance
  
  \[ \text{Imp}_l^2 = \text{Avg} \left[ \sum_{m=1}^{M} \text{Imp}_m I(\text{var}(m) = l) \right] \]
  
  Average overall improvement of objective by variable \( l \)

- Partial Dependence
  
  \[ pd(x_l) = E_{not l} [F(x_l, x_{not l})] \]
  
  Predicted outcome using \( x_l \), after averaging out the others