Parties, Models, Mobsters

A New Implementation of Model-Based Recursive Partitioning in R

Achim Zeileis, Torsten Hothorn

http://eeecon.uibk.ac.at/~zeileis/

Overview

- Model-based recursive partitioning
  - A generic approach
  - Example: Bradley-Terry trees
- Implementation in R
  - Building blocks: Parties, models, mobsters
  - Old implementation in *party*
  - All new implementation in *partykit*
- Illustration

Model-based recursive partitioning

Models: Estimation of parametric models with observations y_i (and regressors x_i), parameter vector θ, and additive objective function Ψ.

$$\widehat{\theta} \quad = \quad \underset{\theta}{\text{argmin}} \sum_i \Psi(y_i, x_i, \theta).$$

Recursive partitioning:

1. Fit the model in the current subsample.
2. Assess the stability of θ across each partitioning variable z_j.
3. Split sample along the z_{j^*} with strongest association: Choose breakpoint with highest improvement of the model fit.
4. Repeat steps 1–3 recursively in the subsamples until some stopping criterion is met.

Model-based recursive partitioning

Parameter instability tests:

- Based on empirical estimating functions (or score/gradient contributions): $\Psi'(y_i, x_i, \hat{\theta})$.
- Under parameter stability: Ψ' fluctuates randomly around its expectation zero.
- Under parameter instability: Systematic departures from zero in subsamples.
- Hence fluctuation can be captured across numeric partitioning variables or within levels of categorical partitioning variables.
- Bonferroni correction for testing across multiple partitioning variables.

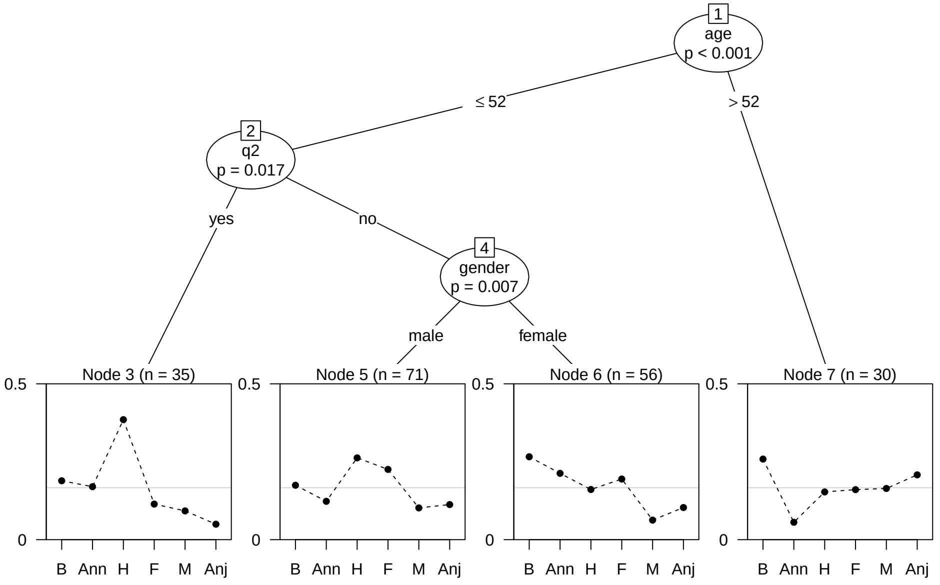
Bradley-Terry trees






Questions: Which of these women is more attractive?
How does the answer depend on age, gender, and the familiarity with the associated TV show Germany's Next Topmodel?

Bradley-Terry trees



Implementation: Building blocks

Workhorse function: `mob()` for

- data handling,
- calling model fitters,
- carrying out parameter instability tests and
- recursive partitioning algorithm.

Required functionality:

- *Parties:* Class and methods for recursive partytions.
- *Models:* Fitting functions for statistical models (optimizing suitable objective function).
- *Mobsters:* High-level interfaces (`lmtree()`, `bttree()`, `...`) that call lower-level `mob()` with suitable options and methods.

Implementation: Old `mob()` in *party*

Parties: S4 class '`BinaryTree`'.

- Originally developed only for `ctree()` and somewhat "abused".
- Rather rigid and hard to extend.

Models: S4 '`StatModel`' objects.

- Intended to conceptualize unfitted model objects.
- Required some "glue code" to accomodate non-standard interface for data handling and model fitting.

Mobsters:

- `mob()` already geared towards (generalized) linear models.
- Other interfaces in *psychotree* and *betareg*.
- Hard to do fine control due to adopted S4 classes: Many unnecessary computations and copies of data.

Implementation: New `mob()` in *partykit*

Parties: S3 class '`modelparty`' built on '`party`'.

- Separates data and tree structure.
- Inherits generic infrastructure for printing, predicting, plotting, ...

Models: Plain functions with input/output convention.

- Basic and extended interface for rapid prototyping and for speeding up computings, respectively.
- Only minimial glue code required if models are well-designed.

Mobsters:

- `mob()` completely agnostic regarding models employed.
- Separate interfaces `lmtree()`, `glmtree()`, ...
- New interfaces typically need to bring their model fitter and adapt the main methods `print()`, `plot()`, `predict()` etc.

Implementation: New `mob()` in *partykit*

New inference options: Not used by default by optionally available.

- New parameter instability tests for ordinal partitioning variables. Alternative to unordered χ^2 test but computationally intensive.
- Post-pruning based on information criteria (e.g., AIC or BIC), especially for very large datasets where traditional significance levels are not useful.
- Multiway splits for categorical partitioning variables.
- Treat weights as proportionality weights and not as case weights.

Implementation: Models

Input: Basic interface.

```
fit(y, x = NULL, start = NULL, weights = NULL,
  offset = NULL, ...)
```

`y`, `x`, `weights`, `offset` are (the subset of) the preprocessed data.
Starting values and further fitting arguments are in `start` and `...`.

Output: Fitted model object of class with suitable methods.

- `coef()`: Estimated parameters $\hat{\theta}$.
- `logLik()`: Maximized log-likelihood function $-\sum_i \Psi(y_i, x, \hat{\theta})$.
- `estfun()`: Empirical estimating functions $\Psi'(y_i, x_i, \hat{\theta})$.

Implementation: Models

Input: Extended interface.

```
fit(y, x = NULL, start = NULL, weights = NULL,
  offset = NULL, ..., estfun = FALSE, object = FALSE)
```

Output: List.

- `coefficients`: Estimated parameters $\hat{\theta}$.
- `objfun`: Minimized objective function $\sum_i \Psi(y_i, x_, \hat{\theta})$.
- `estfun`: Empirical estimating functions $\Psi'(y_i, x_i, \hat{\theta})$. Only needed if `estfun = TRUE`, otherwise optionally `NULL`.
- `object`: A model object for which further methods could be available (e.g., `predict()`, or `fitted()`, etc.). Only needed if `object = TRUE`, otherwise optionally `NULL`.

Internally: Extended interface constructed from basic interface if supplied. Efficiency can be gained through extended approach.

Illustration: Bradley-Terry trees

Data, packages, and `estfun()` method:

```
R> data("Topmodel2007", package = "psychotree")
R> library("partykit")
R> library("psychotools")
R> estfun.btReg <- function(x, ...) x$estfun
```

Basic model fitting function:

```
R> btfit1 <- function(y, x = NULL, start = NULL, weights = NULL,
+    offset = NULL, ...) btReg.fit(y, weights = weights, ...)
```

Fit Bradley-Terry tree:

```
R> system.time(bt1 <- mob(
+    preference ~ 1 | gender + age + q1 + q2 + q3,
+    data = Topmodel2007, fit = btfit1))

   user  system elapsed
  5.112   0.020   5.263
```

Illustration: Bradley-Terry trees

Extended model fitting function:

```
R> btfit2 <- function(y, x = NULL, start = NULL, weights = NULL,
+    offset = NULL, ..., estfun = FALSE, object = FALSE) {
+    rval <- btReg.fit(y, weights = weights, ...,
+      estfun = estfun, vcov = object)
+    list(
+      coefficients = rval$coefficients,
+      objfun = -rval$loglik,
+      estfun = if(estfun) rval$estfun else NULL,
+      object = if(object) rval else NULL
+    )
+  }
```

Fit Bradley-Terry tree again:

```
R> system.time(bt2 <- mob(
+    preference ~ 1 | gender + age + q1 + q2 + q3,
+    data = Topmodel2007, fit = btfit2))

   user  system elapsed
  4.004   0.012   4.087
```

Illustration: Bradley-Terry trees

```
Model-based recursive partitioning (btfit2)

Model formula:
preference ~ 1 | gender + age + q1 + q2 + q3

Fitted party:
[1] root
|   [2] age <= 52
|   |   [3] q2 in yes: n = 35
|   |       Barbara    Anni    Hana   Fiona   Mandy
|   |        1.3378  1.2318  2.0499  0.8339  0.6217
|   |   [4] q2 in no
|   |   |   [5] gender in male: n = 71
|   |   |        Barbara     Anni     Hana    Fiona    Mandy
|   |   |        0.43866  0.08877  0.84629  0.69424 -0.10003
|   |   |   [6] gender in female: n = 56
|   |   |       Barbara    Anni    Hana   Fiona   Mandy
|   |   |        0.9475  0.7246  0.4452  0.6350 -0.4965
|   [7] age > 52: n = 30
|       Barbara    Anni    Hana   Fiona   Mandy
|        0.2178 -1.3166 -0.3059 -0.2591 -0.2357
```

Illustration: Bradley-Terry trees

```
Number of inner nodes:    3
Number of terminal nodes: 4
Number of parameters per node: 5
Objective function: 1829
```

Standard methods readily available:

```
R> plot(bt2)
R> coef(bt2)
```

```
  Barbara     Anni    Hana   Fiona   Mandy
3  1.3378  1.23183  2.0499  0.8339  0.6217
5  0.4387  0.08877  0.8463  0.6942 -0.1000
6  0.9475  0.72459  0.4452  0.6350 -0.4965
7  0.2178 -1.31663 -0.3059 -0.2591 -0.2357
```

Customization:

```
R> worthf <- function(info) paste(info$object$labels,
+    format(round(worth(info$object), digits = 2)), sep = ": ")
R> plot(bt2, FUN = worthf)
```

Illustration: Bradley-Terry trees

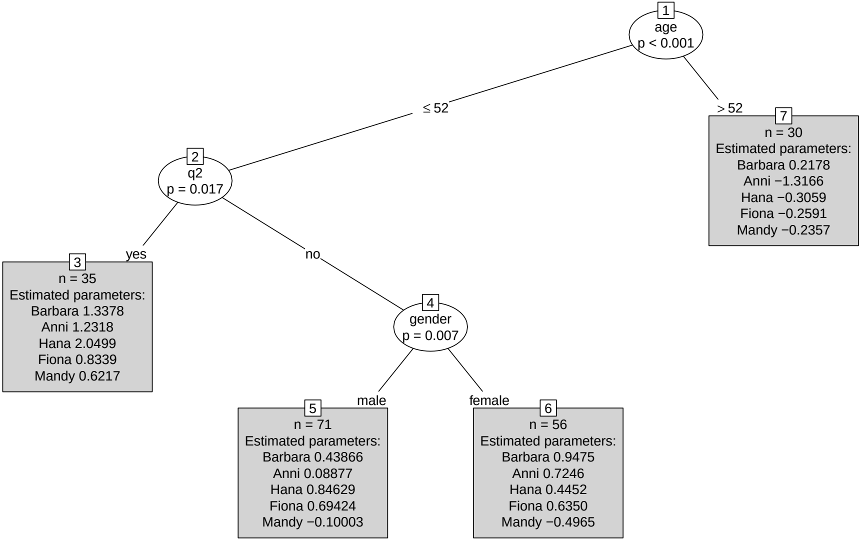


Illustration: Bradley-Terry trees

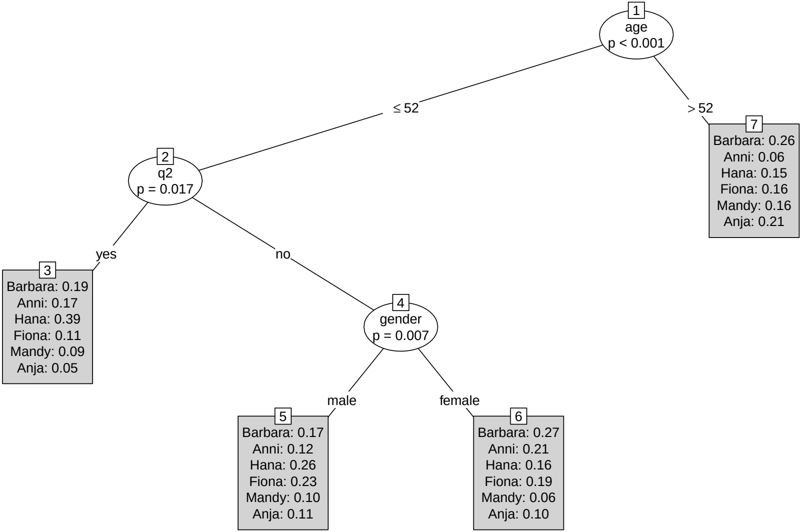


Illustration: Bradley-Terry trees

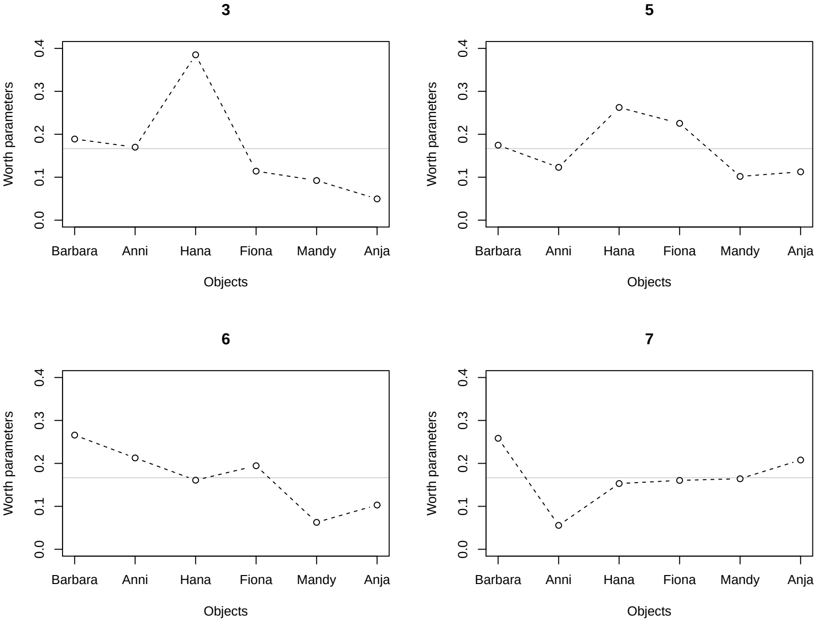


Illustration: Bradley-Terry trees

Apply plotting in all terminal nodes:

```
R> par(mfrow = c(2, 2))
R> nodeapply(bt2, ids = c(3, 5, 6, 7), FUN = function(n)
+    plot(n$info$object, main = n$id, ylim = c(0, 0.4)))
```

Predicted nodes and ranking:

```
R> tm

  age gender q1  q2 q3
1  60   male no  no no
2  25 female no  no no
3  35 female no yes no

R> predict(bt2, tm, type = "node")

1 2 3
7 3 5

R> predict(bt2, tm, type = function(object) t(rank(-worth(object))))

  Barbara Anni Hana Fiona Mandy Anja
1       1    6    5     4     3    2
2       2    3    1     4     5    6
3       3    4    1     2     6    5
```

Summary

- All new implementation of model-based recursive partitioning in *partykit*.
- Enables more efficient computations, rapid prototyping, flexible customization.
- Some new inference options.

References

Hothorn T, Zeileis A (2014). *partykit: A Toolkit for Recursive Partytioning.* R package version 0.2-0. URL `http://R-Forge.R-project.org/projects/partykit/`

Zeileis A, Hothorn T (2014). *Parties, Models, Mobsters: A New Implementation of Model-Based Recursive Partitioning in R.* `vignette("mob", package = "partykit")`.

Strobl C, Wickelmaier F, Zeileis A (2011). "Accounting for Individual Differences in Bradley-Terry Models by Means of Recursive Partitioning." *Journal of Educational and Behavioral Statistics*, **36**(2), 135–153. `doi:10.3102/1076998609359791`

Zeileis A, Hothorn T, Hornik K (2008). "Model-Based Recursive Partitioning." *Journal of Computational and Graphical Statistics*, **17**(2), 492–514. `doi:10.1198/106186008X319331`