Model-based boosting in \textit{R}

Package \texttt{mboost}

Benjamin Hofner

\texttt{benjamin.hofner@imbe.med.uni-erlangen.de}

Institut für Medizininformatik, Biometrie und Epidemiologie (IMBE)
Friedrich-Alexander-Universität Erlangen-Nürnberg

Statistical Computing 2011
Prerequisites

How to get started?

- Start a recent version of R ($\geq 2.9.0$).
- Install package \texttt{mboost}\footnote{1} \footnote{2} \cite{Hothorn:2010a,Hothorn:2010b} [Hothorn et al., 2010a,b]:
  
  \begin{verbatim}
  R> install.packages("mboost",
  + repos = "http://R-Forge.R-project.org")
  \end{verbatim}

- Load package \texttt{mboost}:
  
  \begin{verbatim}
  R> library("mboost")
  \end{verbatim}

\footnote{1} installs current development version of \texttt{mboost}, which is required to have all the features discussed in this tutorial; features \textbf{only} available in \texttt{mboost} $\geq 2.1.0$ are indicated via \footnote{2} to be able to install \texttt{mboost} from R-forge you need to run the \textbf{current} version of R, i.e., R 2.13
glmboost
**glmboost**

- `glmboost()` can be used to fit linear models via component-wise boosting.

- Note that each column of the design matrix is fitted and selected separately using a simple linear model (i.e., linear base-learner).
glmboost(formula, data = list(), weights = NULL, center = TRUE, control = boost_control(), ...)

Only selected options are displayed throughout this talk!

Usage in essence very similar to \texttt{lm()} and \texttt{glm()}.

\textbf{formula}: a formula as in \texttt{lm()}. E.g.:
\begin{verbatim}
formula = y \sim x1 + x2 + x3
\end{verbatim}

\textbf{data}: a data set containing the variables of the formula.

\textbf{weights}: (optional) vector of weights used for weighted regression.

\textbf{center}: a logical variable indicating of the predictor variables are centered before fitting (default = \texttt{TRUE}) (\textgreater{} Slide 7).

\textbf{control}: parameters controlling the algorithm (\textgreater{} Slide 8).

Indirectly available (via "..."; possible parameters see \texttt{?mboost_fit}):

\textbf{family}^{3}: used to specify the fitting problem. For example:
\begin{verbatim}
family = Gaussian()  # default
family = Binomial()  # logit model
\end{verbatim}

^{3}details \textgreater{} see next talks
This is very important (at least for quick convergence).
Each of the columns of the design matrix specifies one base-learner, which is fitted separately to the negative gradient.
The single least-squares fit does not contain an intercept, i.e., models of the type $\text{lm}(u \sim -1 + x1)$ are fitted.
Model is forced through origin $(0, 0)$ which is only reasonable if there are no observations $\triangleright$ centering!
\text{control} = \text{boost\_control}(\text{mstop} = 100, \text{nu} = 0.1, \\
\qquad \text{trace} = \text{FALSE})

\textbf{mstop:} \quad \text{number of } \textbf{initial} \text{ boosting iterations.}

\textbf{nu:} \quad \text{step size } (\in (0, 1])

\textbf{trace:} \quad \text{should status information during the fitting process be printed?}
Example

```r
R> mod.cars <- glmboost(dist ~ speed, data = cars)
R> mod.cars

Generalized Linear Models Fitted via Gradient Boosting

Call:
glmboost.formula(formula = dist ~ speed, data = cars)

Squared Error (Regression)

Loss function: (y - f)^2

Number of boosting iterations: mstop = 100
Step size: 0.1
Offset: 42.98

Coefficients:
(Intercept) speed
-60.557486 3.932304

attr(,"offset")
[1] 42.98
```
\texttt{R> plot(dist ~ speed, data = cars)} \hspace{1cm} \texttt{R> plot(mod.cars)}

\texttt{R> cf <- coef(mod.cars,}
+ \hspace{1cm} \texttt{off2int = TRUE)}

\texttt{R> abline(cf)}

\texttt{R> abline(lm(dist ~ speed, data = cars),}
+ \hspace{1cm} \texttt{col = "gray", lty = 2)}
glmboost with Matrix-Interface

```r
glmboost(x, y, center = TRUE, control = boost_control(), ...)
```

- This is especially useful for high-dimensional data (such as gene expression data).

**x**: (design-) matrix

**y**: response variable (a vector)

**center**: a logical variable indicating of the predictor variables are centered before fitting (default = TRUE) (⊿ Slide 7).

**control**: parameters controlling the algorithm (⊿ Slide 8).
Alternative specification using `paste()`

- An alternative specification via the formula interface using `paste()` exists. Here a (low-dimensional) example:
  ```r
  R> rhs <- paste(colnames(data[, colnames(data) != "y"]),
                  +                  collapse = " + ")
  R> fm <- as.formula(paste("y ~", rhs))
  R> fm
  y ~ x1 + x2 + x3 + x4
  R> mod <- glmboost(fm, data = data)
  ```

- This is very useful for high-dimensional non-linear models or models containing factors. In these cases, the model matrix is built internally without requiring further input by the user.
gamboost
**gamboost**

- **gamboost()** can be used to fit linear models or (nonlinear) additive models via component-wise boosting.
- Base-learners need to be specified more explicitly.
- In general: interface very similar to **glmboost()**.
`gamboost(formula, data = list(), ...)`

**formula:** a formula specifying the model (see later).

**data:** a data set containing the variables of the formula.

**Indirectly available (via "..."):**

- **weights:** (optional) vector of weights used for weighted regression.
- **control:** parameters controlling the algorithm (▶ Slide 8).
- **family:** used to specify the fitting problem.
Special Base-learners
bols() – ordinary least squares

Allows the definition of (penalized) ordinary least squares base-learners, including

1) linear effects \((x \beta)\)
2) categorical effects \((z^\top \beta)\)
3) linear effects for groups of variables \(x = (x_1, x_2, \ldots, x_p)^\top (x^\top \beta)\)
4) ridge-penalized effects for 2) or 3)
5) special penalty for ordinal variables \((x - \text{as.ordered}(x))\)
6) Varying coefficient terms (i.e., interactions)
7) …
\texttt{bols(..., by = NULL, intercept = TRUE, df = NULL, lambda = 0)}

\textbf{...:} variables are specified here (comma-separated)

\textbf{by:} additional variable that defines varying coefficients (optional)

\textbf{intercept:} if \texttt{intercept = TRUE} an intercept is added to the design matrix. If \texttt{intercept = FALSE}, continuous covariates should be (mean-) centered.

\textbf{df:} initial degrees of freedom for penalized estimates (▷ Slide 42)

\textbf{lambda:} smoothing penalty, computed from \texttt{df} when \texttt{df} specified
Examples for `bols()`

**Linear effect**

True effect:

\[ y = 0.5x_1 \]

**Categorical effect**

True effect:

\[ y = 0x_2^{(1)} - 1x_2^{(2)} + 0.5x_2^{(3)} + 3x_2^{(4)} \]
Some calls using `bols()`

`bols(x)`

linear effect $\triangleright x^\top \beta$ ($x^\top = (1, x)$)

`bols(z)`

OLS fit with factor $z$ (i.e., linear effect after dummy coding)

`bols(x1, x2, x3)`

one base-learner for three variables (group-wise selection)

`bols(x, by = z)`

interaction (with $z$ continuous or factor variable) $\triangleright \beta \cdot xz$

`bols(x, intercept = FALSE)`

linear effect without intercept $\triangleright \beta \cdot x$
bbs() – B-splines (with penalty)

Allows the definition of smooth effects based on B-splines with difference penalty (i.e., P-splines). Examples include:

1) smooth effects \( f(x) \)
2) bivariate smooth effects (e.g., spatial effects \( f(x, y) \))
3) varying coefficient terms \( f(x) \cdot z \)
4) cyclic effects (= periodic effects)
5) . . .
\texttt{bbs(..., by = NULL, knots = 20, boundary.knots = NULL,}
\texttt{degree = 3, differences = 2, df = 4, lambda = NULL,}
\texttt{center = FALSE, cyclic = FALSE)}

\ldots: variables are specified here (comma-separated, \textbf{max. 2 variables}).

\textbf{knots}: number of equidistant knots (a), or positions of the interior knots (b) or a named list (with (a) or (b))

\textbf{boundary.knots} \textsuperscript{2.1-0}: boundary knot locations (default: range of the data)

\textbf{degree}: degree of the B-spline basis

\textbf{differences}: difference order of penalty ($\in \{0, 1, 2, 3\}$)

\textbf{center}: reparameterization of P-splines (\textsuperscript{\textgreater} Slide 24)

\textbf{cyclic} \textsuperscript{2.1-0}: if \texttt{cyclic} = TRUE the fitted values coincide at the boundary knots

\textbf{by}: additional variable that defines varying coefficients (optional)

\textbf{df}: initial degrees of freedom (\textsuperscript{\textgreater} Slide 42)

\textbf{lambda}: smoothing penalty, computed from \texttt{df} when \texttt{df} specified
Examples for \texttt{bbs()}
Some calls using `bbs()`

- `bbs(x, by = z)`
  - varying coefficient $\triangleright f(x) \cdot z = \beta(x)z$

- `bbs(x, knots = 10)`
  - smooth effect with 10 inner knots

- `bbs(x, boundary.knots = (0, 2 * pi), cyclic = TRUE)`
  - cyclic effect, i.e., periodic function with period $2\pi$
    - $\triangleright$ important to specify boundary knots (which are the points where the function is joined)

- `bbs(x, center = TRUE, df = 1)`
  - P-spline decomposition (`center = TRUE`), which is needed to specify arbitrary small `df` for P-splines
    - $\triangleright$ in this case (usually) you need to add `bols(x, intercept = FALSE)` to the model formula
    - $\triangleright$ see Kneib et al. [2009]
Example: Cyclic Splines

True effect: \( f(x) = \sin(x) \)

StatComp 2011 (Benjamin Hofner)
\texttt{bspatial(..., df = 6)}

Allows to specify spatial, smooth effects (via bivariate P-splines). \texttt{bspatial()} is just a wrapper to \texttt{bbs()} with redefined degrees of freedom. Note that \texttt{df = 4} was changed to \texttt{df = 6} in \texttt{mboost} 2.1-0.
Examples for `bspatial()`

True function: $f(x_1, x_2) = \sin(x_1) \cdot \sin(x_2)$
**brandom(...)**

Allows to specify (quasi) random effects (via ridge penalized linear effects). `brandom()` is just a wrapper to `bols()` with redefined degrees of freedom.

- Examples of `brandom()` see below.
Additional Base-learners in a Nutshell

*btree()*\(^{2.1-0}\)  
Tree-based base-learner (per default: stumps)

*brad()*\(^ {2.1-0}\)  
Radial basis functions (e.g., for spatial effects)

*bmono()*\(^ {2.1-0}\)  
Monotonic effects of continuous or ordered categorical variables

*bmrf()*\(^ {2.1-0}\)  
Markov random fields (for spatial effects of regions)

*buser()*\(^ {2.1-0}\)  
User specified base-learner (advanced users only!)
Building a Model

How to combine different base-learners

- **Simple formula:**
  ```r
  R> gamboost(y ~ bols(x1) + bbs(x2) + bspatial(s1, s2) + brandom(id),
              +   data = example)
  ```

- **Options for some base-learners:**
  ```r
  R> gamboost(y ~ bols(int, intercept = FALSE) +
              +   bols(x1, intercept = FALSE) +
              +   bbs(x2) +
              +   bspatial(s1, s2, knots = list(s1 = 10, s2 = 20)) +
              +   brandom(id) + brandom(id, by = x1),
              +   data = example)
  ```

- Separate intercept (where `example$int = rep(1, length(y))`)
- Base-learner for `x1` without intercept (i.e., $x_1 \beta_1$)
- 10 inner knots in `s1` direction, 20 inner knots in `s2` direction, i.e., $10 \cdot 20 = 200$ inner knots in total;
  Usually, something like `knots = 10` (≫ 10 knots per coordinate)
- Random slope for `x1` (in the groups of `id`)
cvrisk and subset method
Optimal stopping iteration

- model needs tuning to prevent overfitting
- determine optimal stopping iteration
- Use cross-validated estimates of the empirical risk to choose an appropriate number of boosting iterations.
- aims at optimizing prognosis on new data
- infrastructure in **mboost** exists to compute
  - bootstrap estimates
  - k-fold cross-validation estimates
  - sub-sampling estimates
  - and to do this in parallel
cvrisk

How to determine \( m_{\text{stop}} \)

```
cvrisk(object, folds = cv(model.weights(object)),
        grid = 1:mstop(object),
        papply = if (require("multicore")) mclapply else lapply)
```

**object:** the mboost model

**folds:** weight matrix that determines the crossvalidation samples
(▷ see `cv()` on next slide)

**grid:** compute empirical (out-of-bag) risk on this grid of \( m_{\text{stop}} \) values.

**papply:** use `mclapply` if `multicore` is available, else run sequentially;
Alternative: `clusterApplyLB` (package `snow`) can be used here (with some further setup needed ▷ see `?cvrisk`).
cv(weights, type = c("bootstrap", "kfold", "subsampling"),
    B = ifelse(type == "kfold", 10, 25))

weights: (original) weights of the model; one can use model.weights(mod) to extract the weights from mod.

type: use bootstrap (default), k-fold cross-validation or sub-sampling

B: number of folds, per default 25 for bootstrap and subsampling and 10 for kfold

▷ returns a matrix with B columns that determines the weights for each cross-validation run
Example

Determine mstop

R> model <- glmboost(DExfat ~ ., data = bodyfat, center = TRUE,
control = boost_control(trace = TRUE))

R> ## 10-fold cross-validation
R> cv10f <- cv(model.weights(model), type = "kfold")
R> cvm <- cvrisk(model, folds = cv10f, papply = lapply)
R> cvm

Cross-validated Squared Error (Regression)

```
glmboost.formula(formula = DExfat ~ ., data = bodyfat, center = TRUE,
control = boost_control(trace = TRUE, mstop = 10))
```

```
 1 2 3 4 5 6
105.22746 89.82040 77.04107 66.23743 58.54920 50.82022
```

(...)

```
97 98 99 100
12.94416 12.93962 12.93397 12.94333
```

Optimal number of boosting iterations: 86
\texttt{R> plot(cvm)}
\texttt{R> mstop(cvm)}

\[1\] 86
Subset

How (not) to get rid of spare boosting steps

To increase or reduce the number of boosting steps for the model `mod`, one can use the indexing / subsetting operator:

```r
mod[i]
```

Attention, non-standard behavior:

- `mod[i]` directly changes the model `mod`
- No need to save `mod` under a new name
- Good news: nothing gets lost!

Example:

- fit an initial model `mod` with `mstop = 100`
- set `mod[10]` sets `mstop = 10` (in the model object)
- increasing `mod[40]` requires no re-computation of the model; internally everything was kept in storage!
- But keep attention: `coef(mod)` now returns the coefficients in the 40th boosting iteration!
Example (ctd.)

Determine mstop

\[
R> model[mstop(cvm)]
\]

Generalized Linear Models Fitted via Gradient Boosting

Call:

\[
\text{glmboost.formula(formula = DEXfat \sim ., data = bodyfat, center = TRUE, control = boost\_control(trace = TRUE))}
\]

Squared Error (Regression)

Loss function: \( (y - f)^2 \)

Number of boosting iterations: mstop = 86
Step size: 0.1
Offset: 30.78282
(...)

Model reduced to 86 steps (before per default 100)
To see that nothing got lost we now increase mstop to 200:

```R
model[200]
```

Final risk: 670.6114

Generalized Linear Models Fitted via Gradient Boosting

Call:
glmboost.formula(formula = DEXfat ~ ., data = bodyfat, center = TRUE, control = boost_control(trace = TRUE))

Squared Error (Regression)

Loss function: \((y - f)^2\)

Number of boosting iterations: mstop = 200
Step size: 0.1
Offset: 30.78282

(continues at step 101! (remember that we initially fitted 100 steps)
Summary

Key Messages

mboost ...

... has a modular nature.
... is relatively easy to use.
... offers a well tested back-end.
... is able to specify a wide range of possible models (more to come!).
... has a rich infrastructure (e.g., plot functions, cvrisk, mod[i], ...
... ; more to come!)

Comments

on bugs, on improvements in the manual, etc.
are always welcome (▷ email us!)
References


Controlling smoothness

Degrees of Freedom ↔ Smoothing Parameter

- Degrees of freedom $df$ and smoothing parameter $\lambda$ have a one-to-one relationship.
- **However:** Multiple definitions for degrees of freedom exist
- Let $S = X(X^\top X + \lambda K)^{-1}X^\top$ be the smoother matrix.
  - Standard definition
    \[
    df = \text{trace}(S)
    \] (1)
  - Alternative definition
    \[
    df = \text{trace}(2S - S^\top S)
    \] (2)
- Use
  \[
  R> \text{options(mboost_dftraceS = FALSE)}
  \]
to switch to definition (2) (default: TRUE).
- Definition (2) more suitable for boosting [see Hofner et al., 2010].

**NB:** Define **equal** degrees of freedom (with Def. (2)) for all base-learners to obtain “less biased” selection.