Statistical boosting using the R package `mboost`

Sarah Brockhaus

University of Mannheim, LMU Munich
sarah.brockhaus@mzes.uni-mannheim.de

Thanks to Benjamin Hofner and Andreas Mayr for material

19. October 2016
Social Science Data Lab

Outline

(1) Statistical Boosting

Applying boosting methods to fit statistical models.

(2) The R package `mboost`

Model fitting with the R package `mboost`.

(1) Statistical boosting

Boosting? – searching a dictionary

boost

verb /bu:st/[T]

---

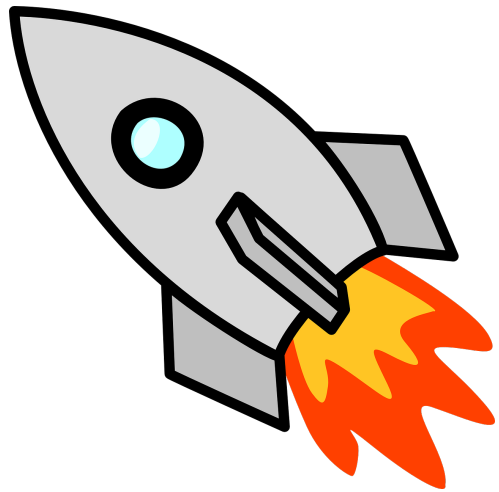
to improve or increase something

> *The theatre managed to boost its audiences by cutting ticket prices.*
>
> [...]
>
> *I tried to boost his ego* (= make him feel more confident) *by praising his cooking.*

(Definition of the verb boost from the Cambridge Advanced Learner's Dictionary & Thesaurus, Cambridge University Press, 2012)

Boosting

- Idea of boosting emerged from the field of machine learning;
  originally as algorithm for classification.
- Boosting was further developed to fit statistical models like LMs, GLMs, GAMs, quantile regression, survival models, ...



(see Mayr et al. (2014) for an overview of the history of boosting algorithms)

Boosting linear models (LMs)

Here, we focus on estimating LMs;

$$\mathbb{E}(y_i) = f(\boldsymbol{x}_i) = h_1(x_{i1}) + \ldots + h_p(x_{ip}), \quad i = 1, \ldots, n$$

- response y_i, covariates $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{ip})^\top$,
- linear predictor $f(\boldsymbol{x}_i)$, and
- covariate effects $h_1(x_{i1}), \ldots, h_p(x_{ip})$

Basic ideas of boosting (1)

- Boosting can fit LMs, GLMs, quantile regression models, ..., by optimizing the corresponding loss function, e.g.,
  - minimizing the L_2-loss (squared loss) yields mean regression (LM / ordinary least squares);
  - minimizing the L_1-loss (absolute loss) yields median regression
  - minimizing the negative log-likelihood yields GLMs

$\rightarrow$ flexible tool to estimate different kinds of regression models

Basic ideas of boosting (2)

Covariate effects: base-learners

- Each covariate effect is represented as a simple model
- The simple models are called base-learners
- Possible base-learners are
  - LMs which yield a linear effect $h_j(x_{ij}) = \beta_j x_{ij}$
  - non-linear models, represented as spline basis with penalty, which yield smooth effects $h_j(x_{ij}) = f_j(x_{ij})$
- Each base-learner is fitted separately (divide and conquer strategy)

$\rightarrow$ the model is composed of many base-learners

$\rightarrow$ flexible tool to estimate models with different covariate effects

Basic ideas of boosting (3)

Boosting is an iterative algorithm:

- Start with an 'empty model' (e.g., the global mean for mean regression)
- In each boosting-step: choose the best-fitting base-learner and update the model accordingly by a small step

$\rightarrow$ With more boosting-steps, the model gets more complex.

$\rightarrow$ Use the number of boosting iterations m_{stop} as tuning parameter.

L_2-Boosting (1)

Fit the LM: $\mathbb{E}(y_i) = f(\boldsymbol{x}_i) = h_1(x_{i1}) + \ldots + h_p(x_{ip}), \quad i = 1, \ldots, n$

Initialization

(1) Set $m := 0$. Initialize the additive predictor $\hat{f}^{[0]} = 0$ (or $\hat{f}^{[0]} = \bar{y}$) and specify a set of base-learners $h_1(x_1), ..., h_p(x_p)$

Fit the residuals

(2) Set $m := m + 1$; compute the residuals using the additive predictor from the previous iteration:

$$u_i^{[m]} = y_i - \hat{f}^{[m-1]}(\boldsymbol{x}_i), \quad i = 1, ..., n$$

(3) Fit each base-learner separately to the current residuals $u_i^{[m]}$:

$$\mathbb{E}(u_i^{[m]}) = \hat{h}_1^{[m]}(x_{i1}), \text{ fit 1st base-learner}$$

$$\vdots$$

$$\mathbb{E}(u_i^{[m]}) = \hat{h}_p^{[m]}(x_{ip}), \text{ fit } p\text{th base-learner}$$

L_2-Boosting (2)

Fit the LM: $\mathbb{E}(y_i) = f(\boldsymbol{x}_i) = h_1(x_{i1}) + \ldots + h_p(x_{ip}), \quad i = 1, \ldots, n$

$\ldots$

Update the best fitting base-learner

(4) Select the base-learner h_{j^*} that best fits $u_i^{[m]}$:

$$j^* = \underset{1 \leq j \leq p}{\operatorname{argmin}} \sum_{i=1}^{n} \left(u_i^{[m]} - \hat{h}_j^{[m]}(x_{ij}) \right)^2 .$$

(5) Update the additive predictor $\hat{f}$ with this base-learner:

$$\hat{f}^{[m]}(\boldsymbol{x}_i) = \hat{f}^{[m-1]}(\boldsymbol{x}_i) + \nu_{\text{sl}} \cdot \hat{h}_{j^*}^{[m]}(x_{ij^*}) ,$$

where ν_{sl} is a small step-length, e.g., $\nu_{\text{sl}} = 0.1$.

Iteration

Iterate steps (2) to (5) until $m = m_{\text{stop}}$.

Some comments on boosting

- boosting can fit models with more parameters than observations ('$p \gg n$')
- base-learners determine the potential covariate effects

- other loss functions than L_2-loss are possible yielding, e.g., GLMs or quantile regression; in general the negative gradient is computed – for the L_2-loss this corresponds to the residuals
- boosting as functional gradient descent – optimization along the steepest gradient descent

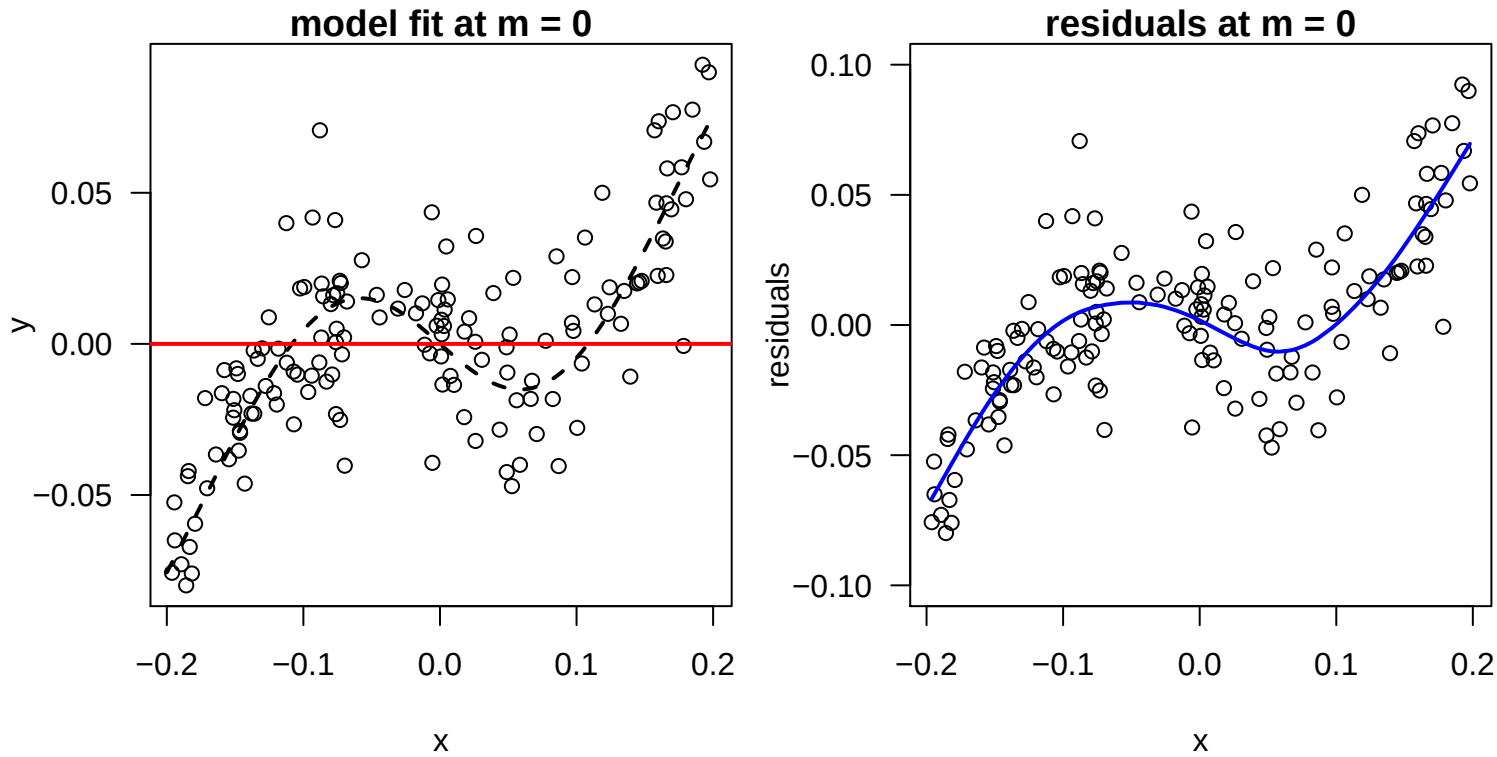
- the most important tuning parameter is the number of boosting iterations m_{stop} as it controls model complexity

Tuning m_{stop}: Variable selection and shrinkage

- Main tuning parameter is the stopping iteration m_{stop}.
  It controls *variable selection* and the *amount of shrinkage*.
  - variables that are never selected in a boosting-step are excluded.
  - boosting shrinkes the effects towards zero (compare to LASSO), leading to more stable predictions.
- For large m_{stop} boosting converges to the same solution as conventional ML algorithms.
- Use resampling methods (cross-validation, bootstrap) to select the m_{stop} that optimizes the predictive risk.

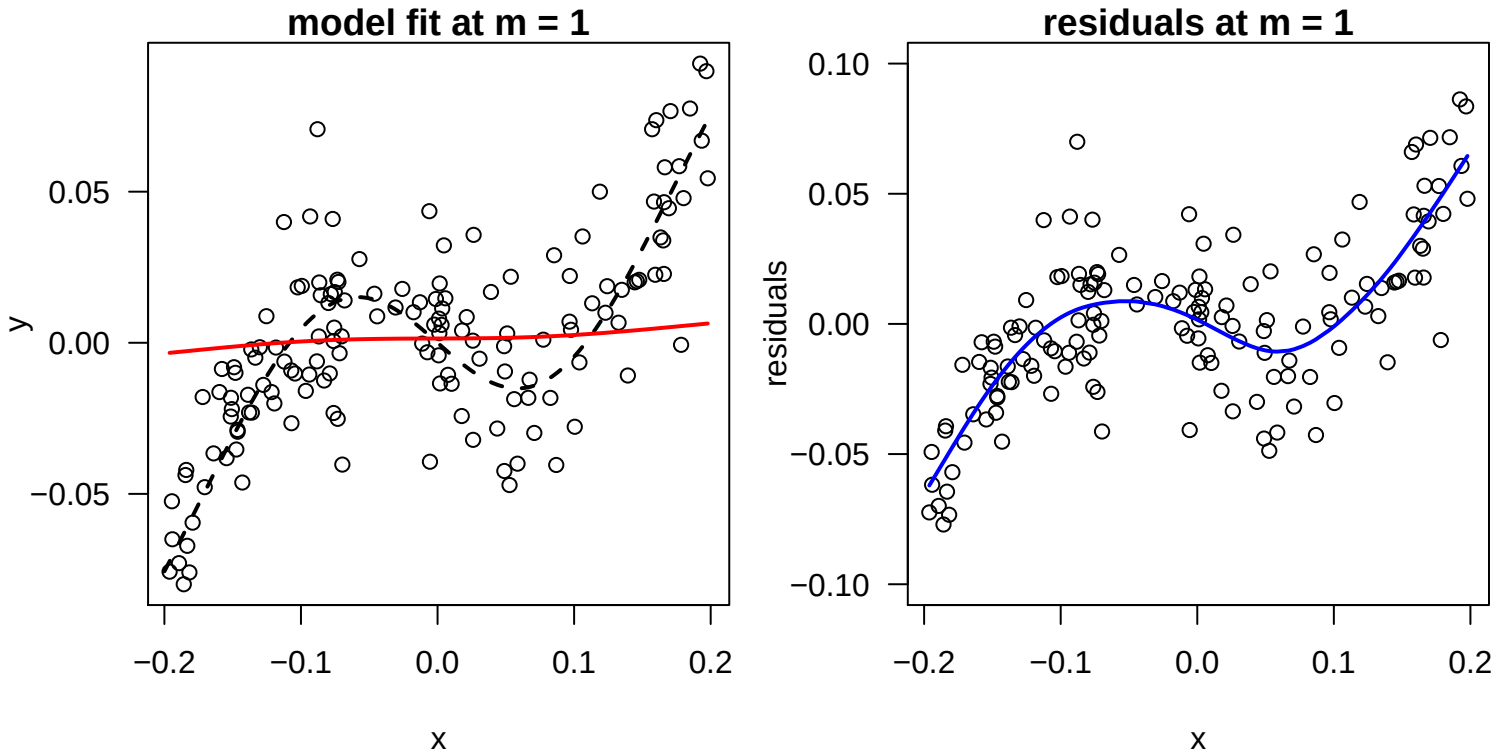
Example for the effect of m_{stop}

Fit an additive model $\mathbb{E}(y_i) = f(x_i)$



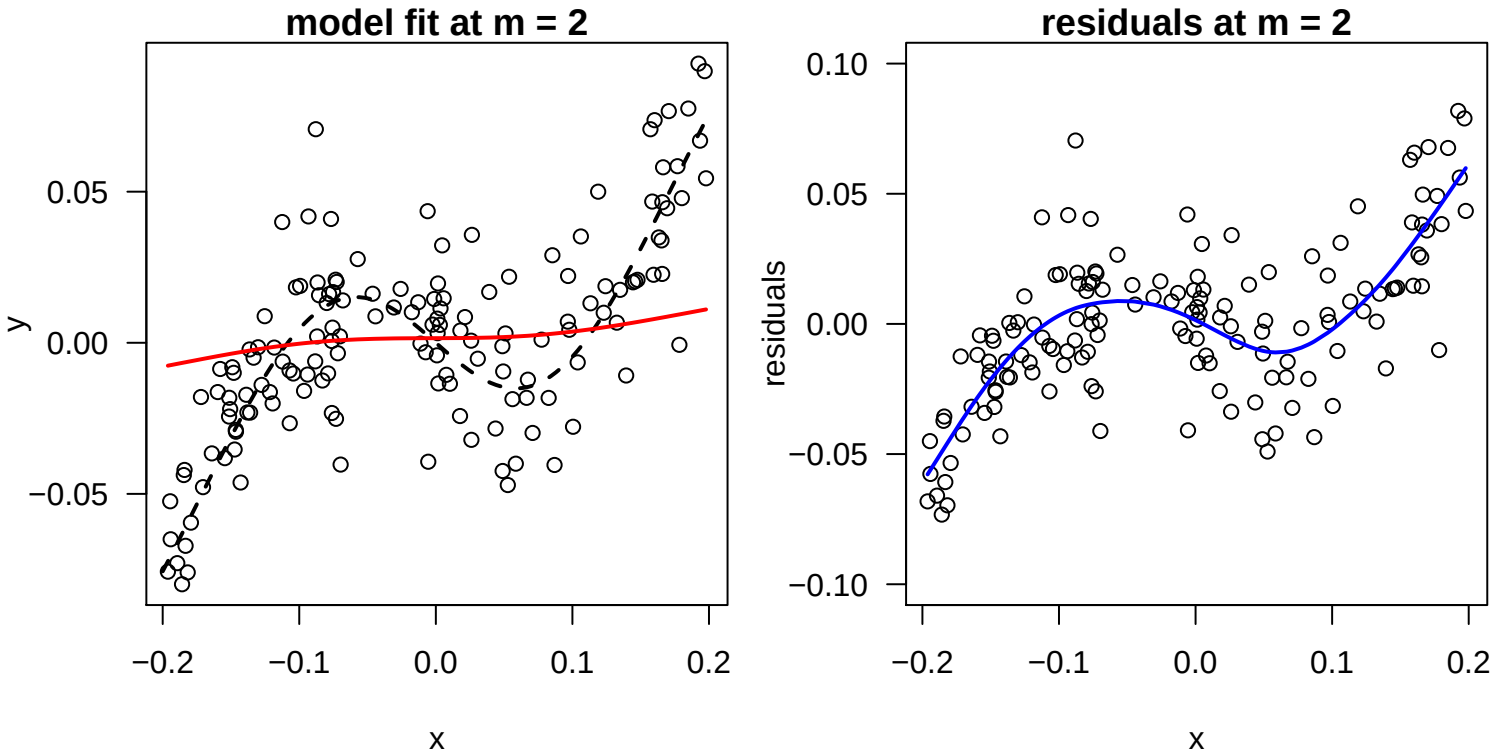
Example for the effect of m_{stop}

Fit an additive model $\mathbb{E}(y_i) = f(x_i)$



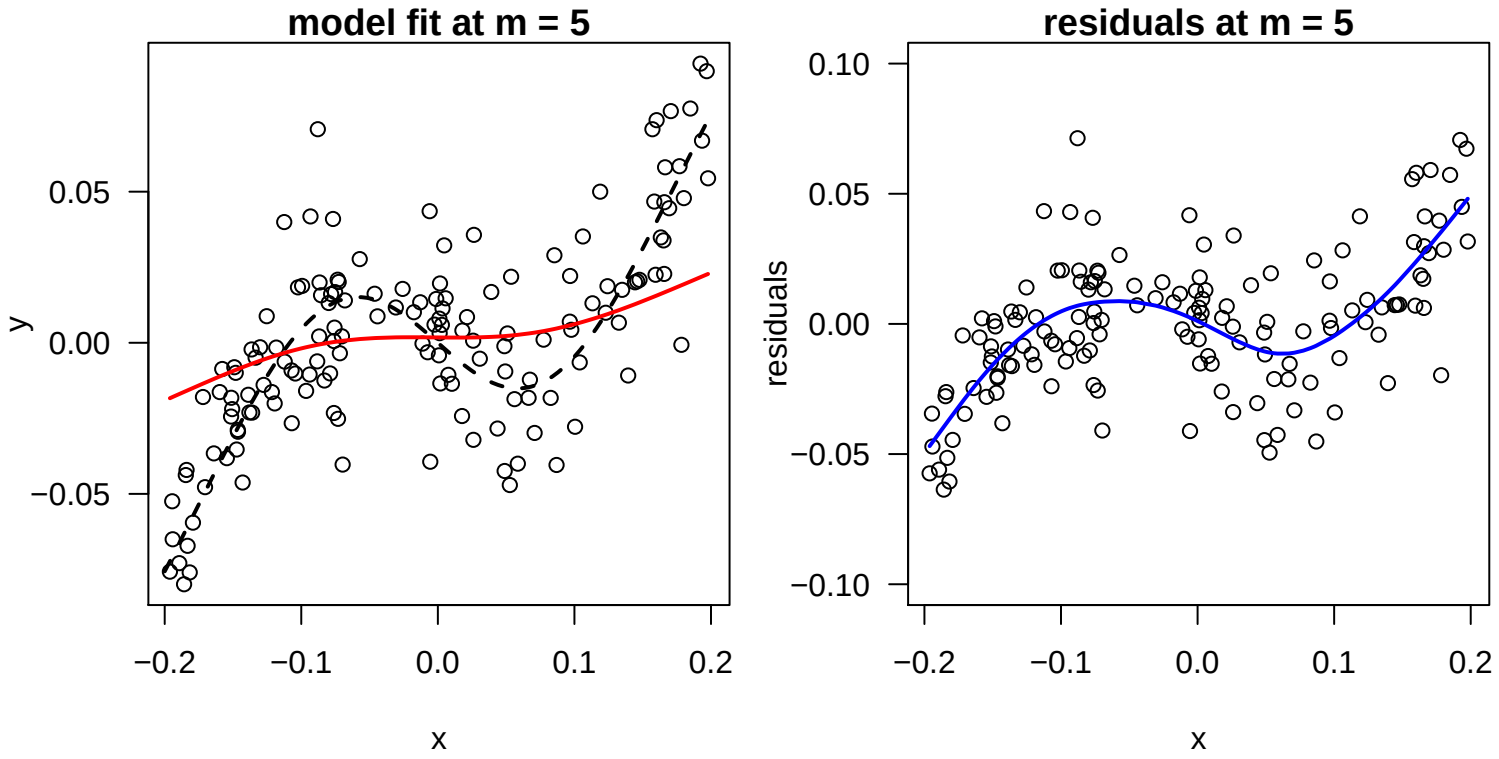
Example for the effect of m_{stop}

Fit an additive model $\mathbb{E}(y_i) = f(x_i)$



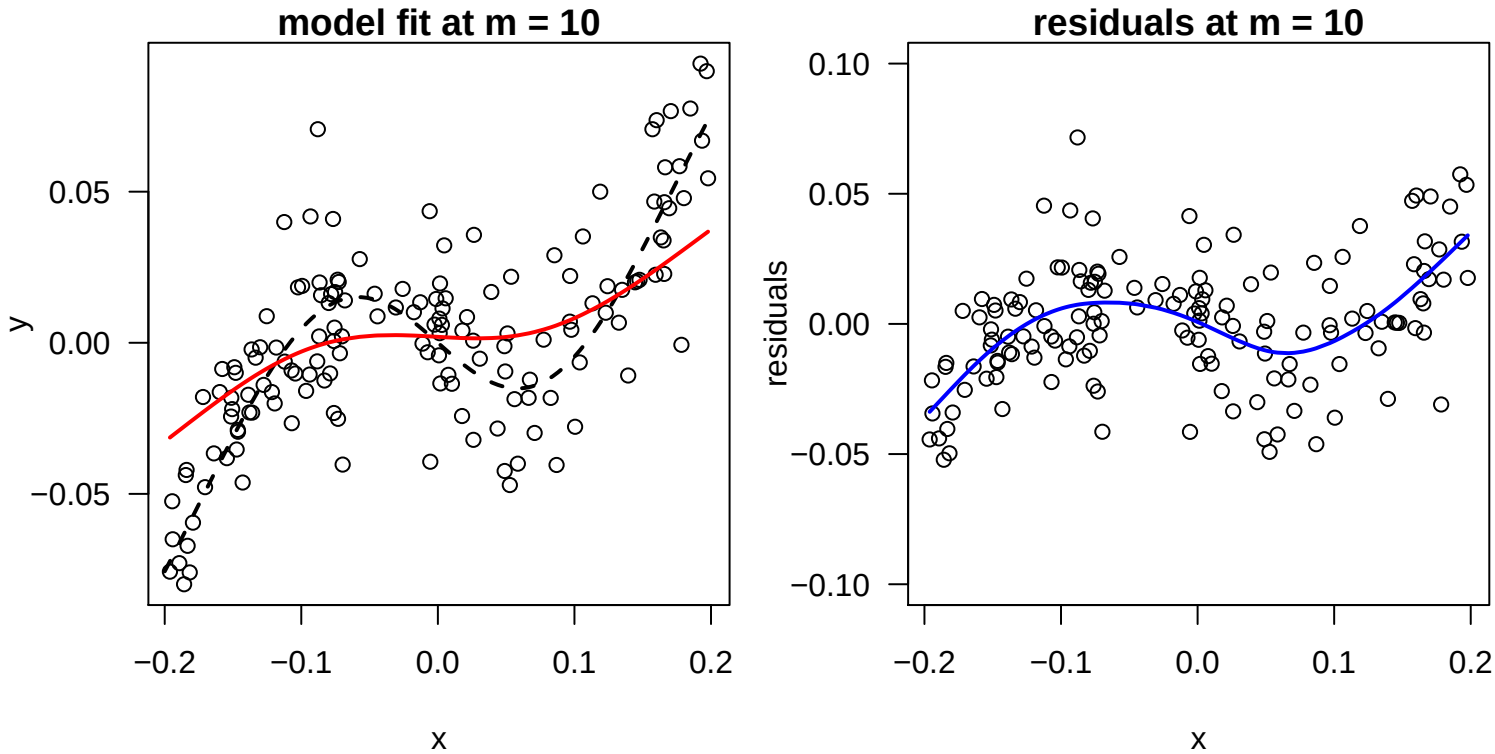
Example for the effect of m_{stop}

Fit an additive model $\mathbb{E}(y_i) = f(x_i)$



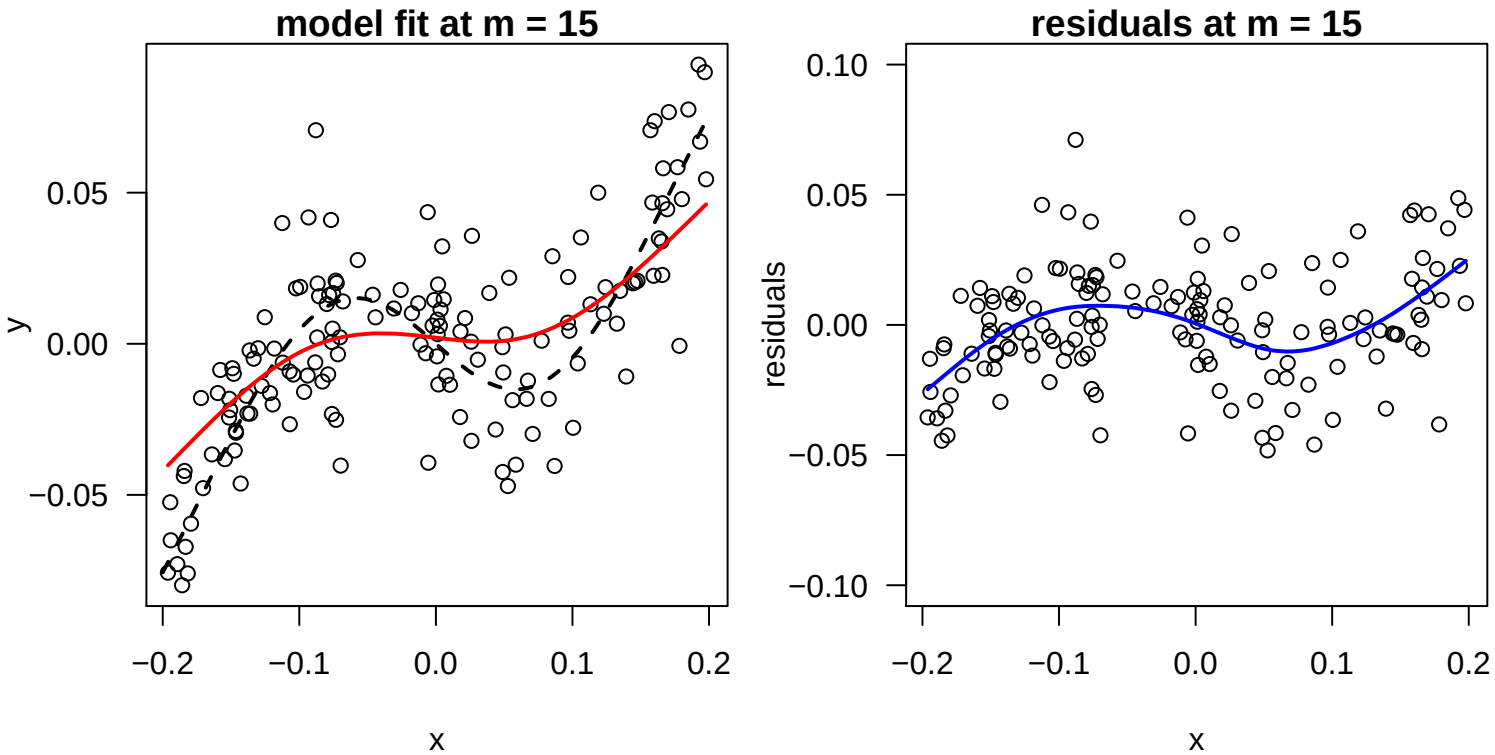
Example for the effect of m_{stop}

Fit an additive model $\mathbb{E}(y_i) = f(x_i)$



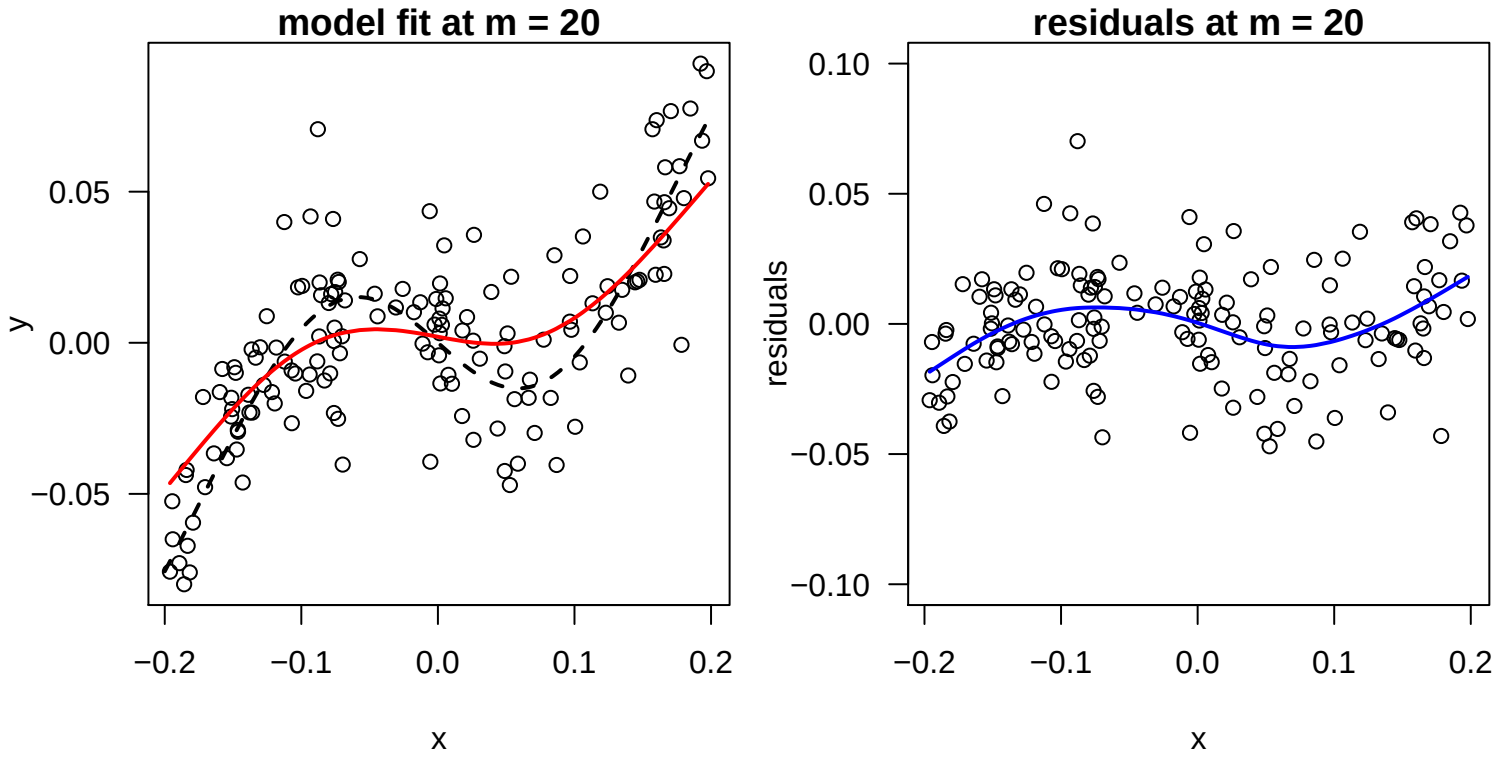
Example for the effect of m_{stop}

Fit an additive model $\mathbb{E}(y_i) = f(x_i)$



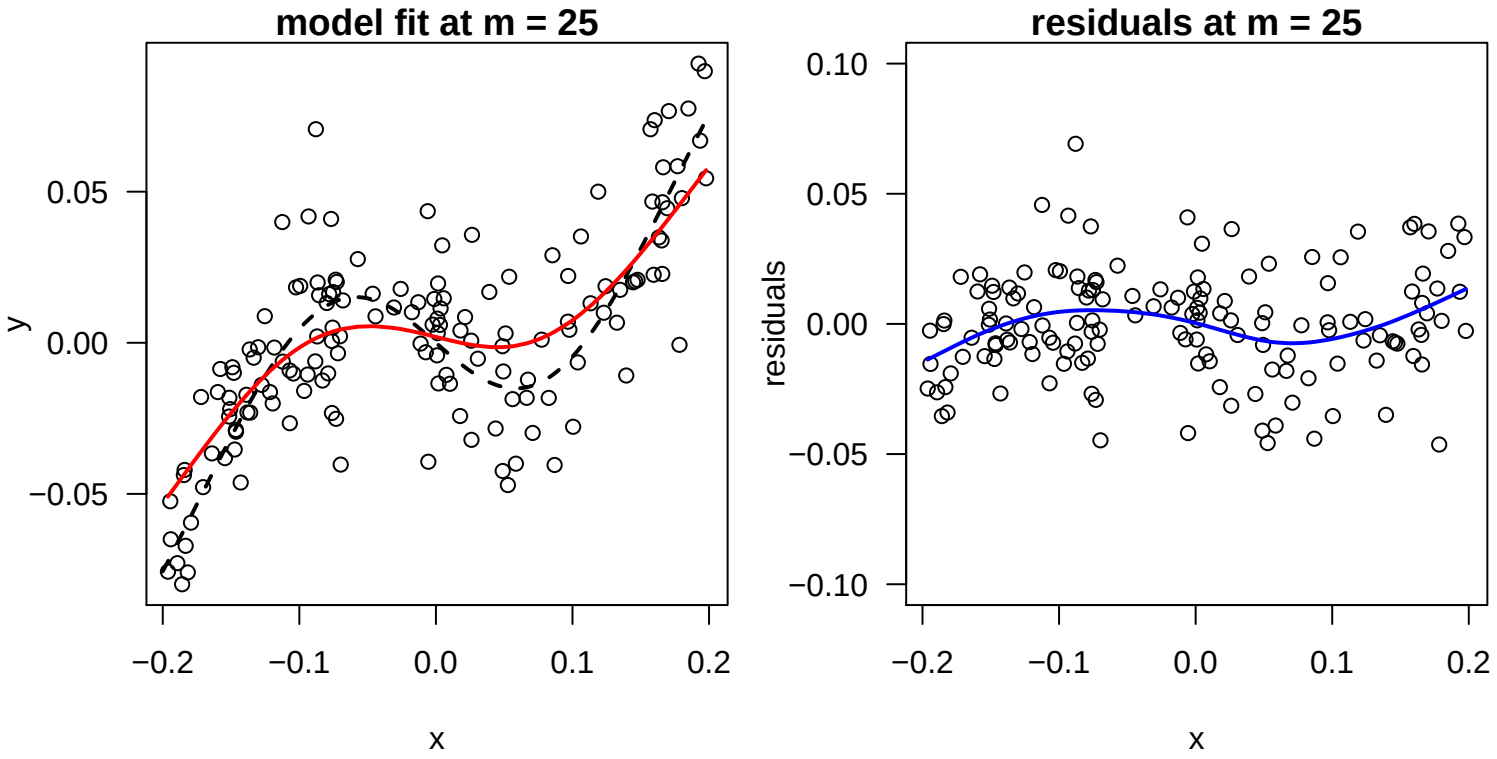
Example for the effect of m_{stop}

Fit an additive model $\mathbb{E}(y_i) = f(x_i)$



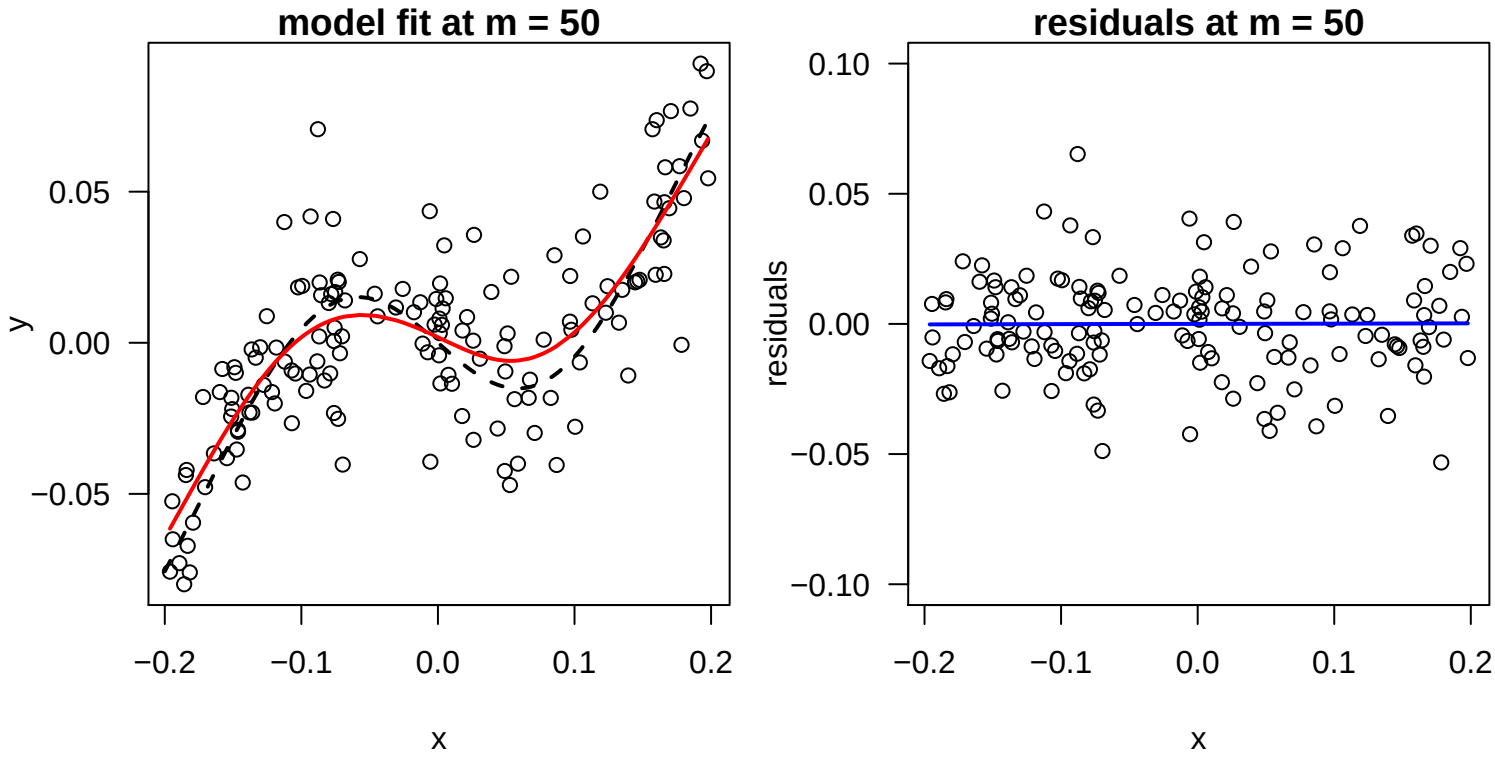
Example for the effect of m_{stop}

Fit an additive model $\mathbb{E}(y_i) = f(x_i)$



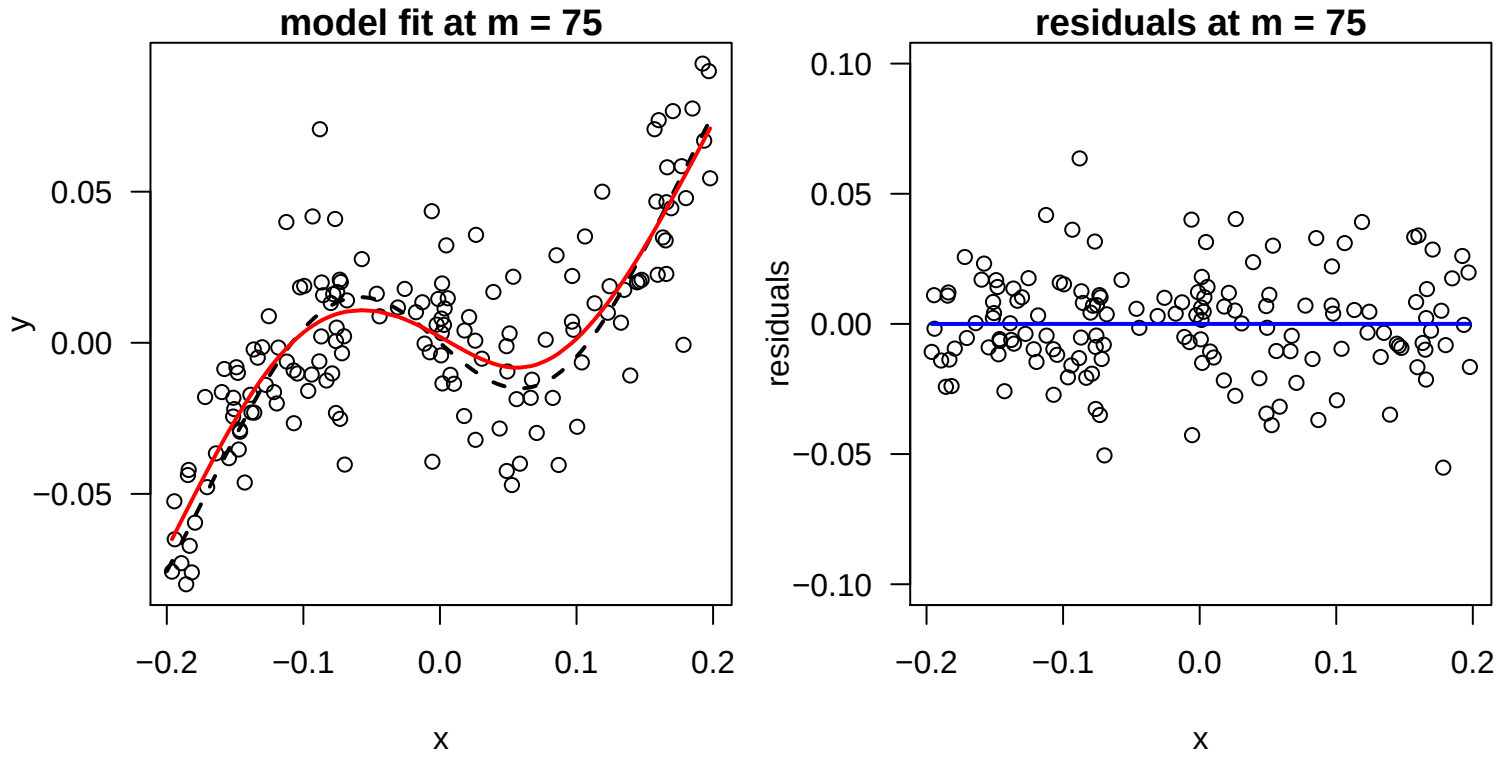
Example for the effect of m_{stop}

Fit an additive model $\mathbb{E}(y_i) = f(x_i)$



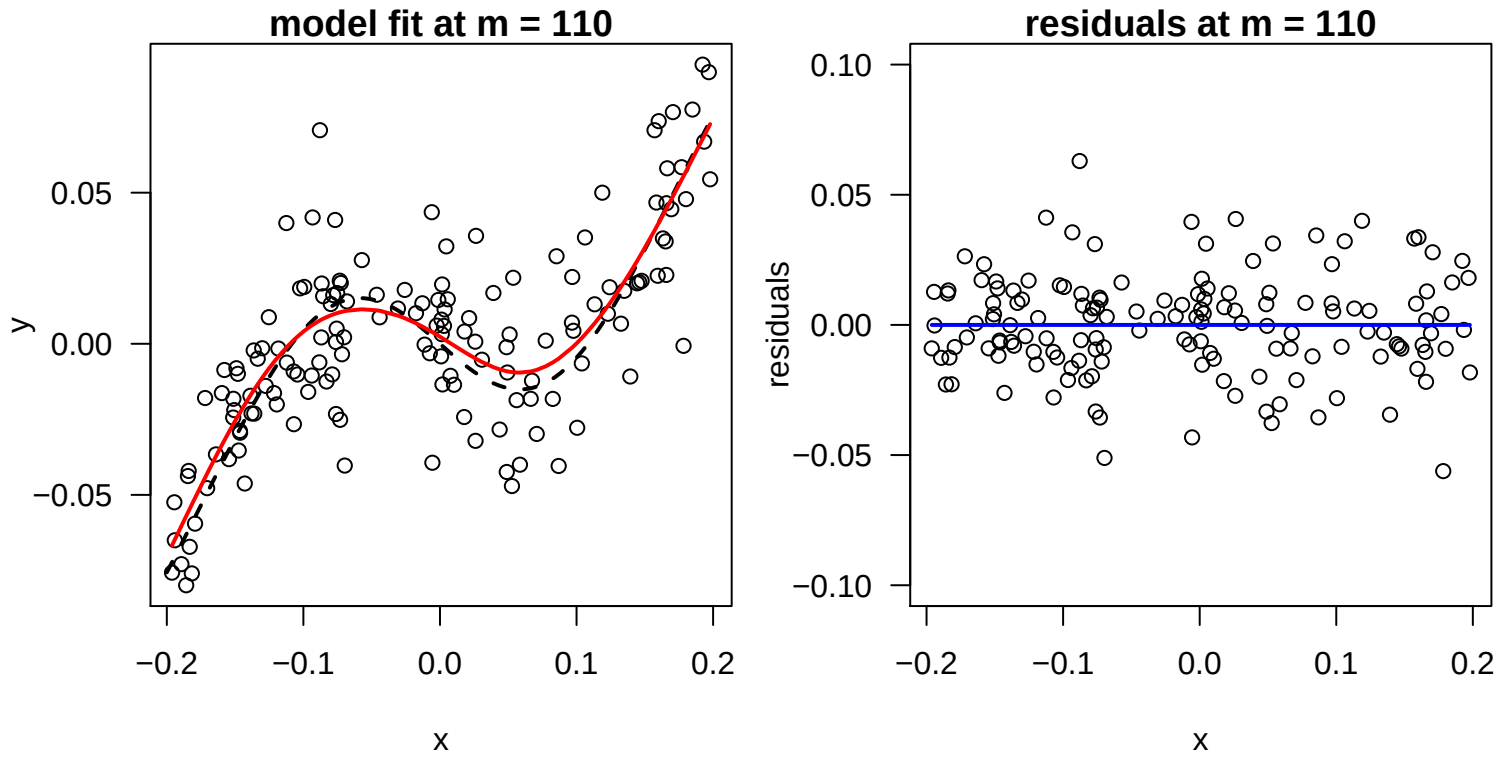
Example for the effect of m_{stop}

Fit an additive model $\mathbb{E}(y_i) = f(x_i)$



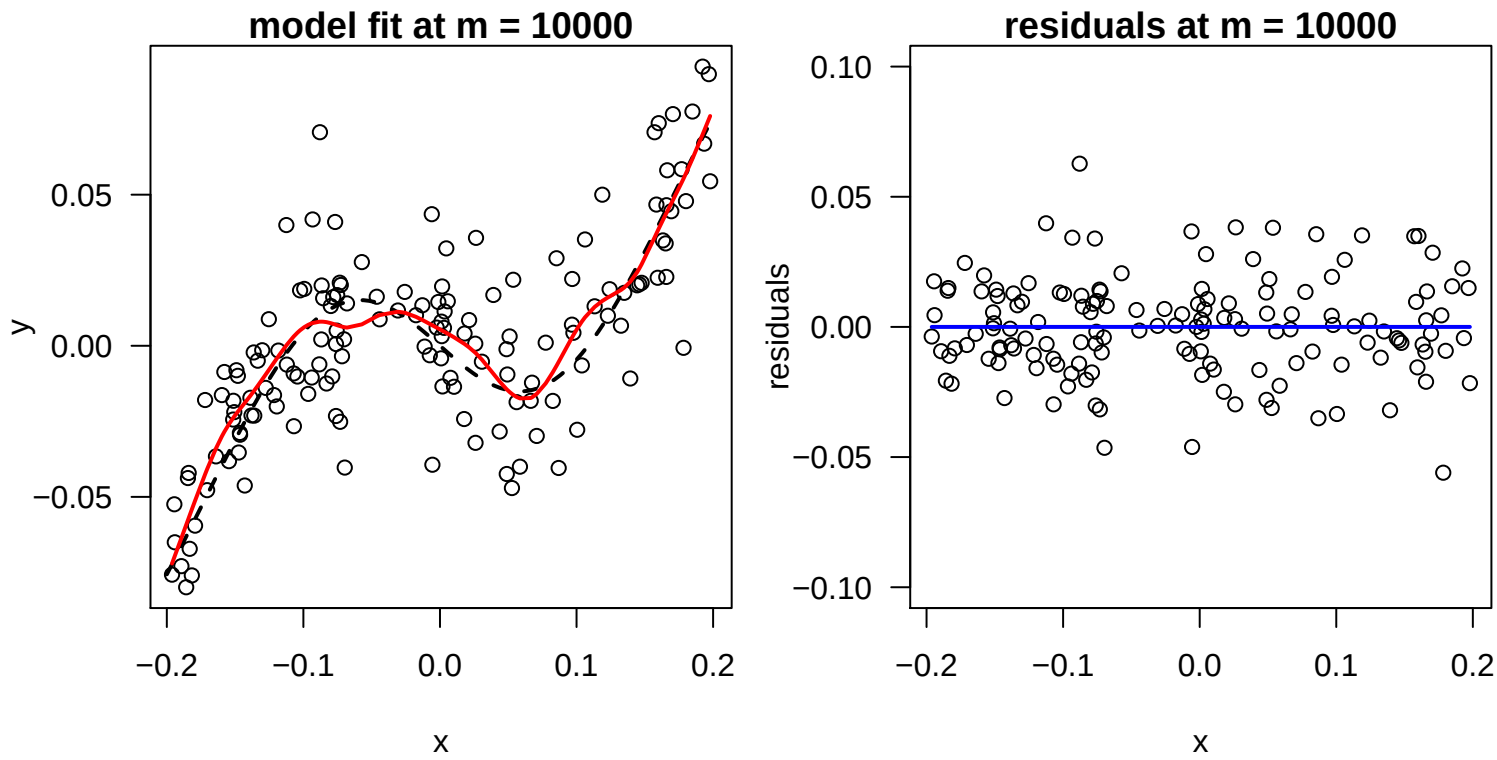
Example for the effect of m_{stop}

Fit an additive model $\mathbb{E}(y_i) = f(x_i)$



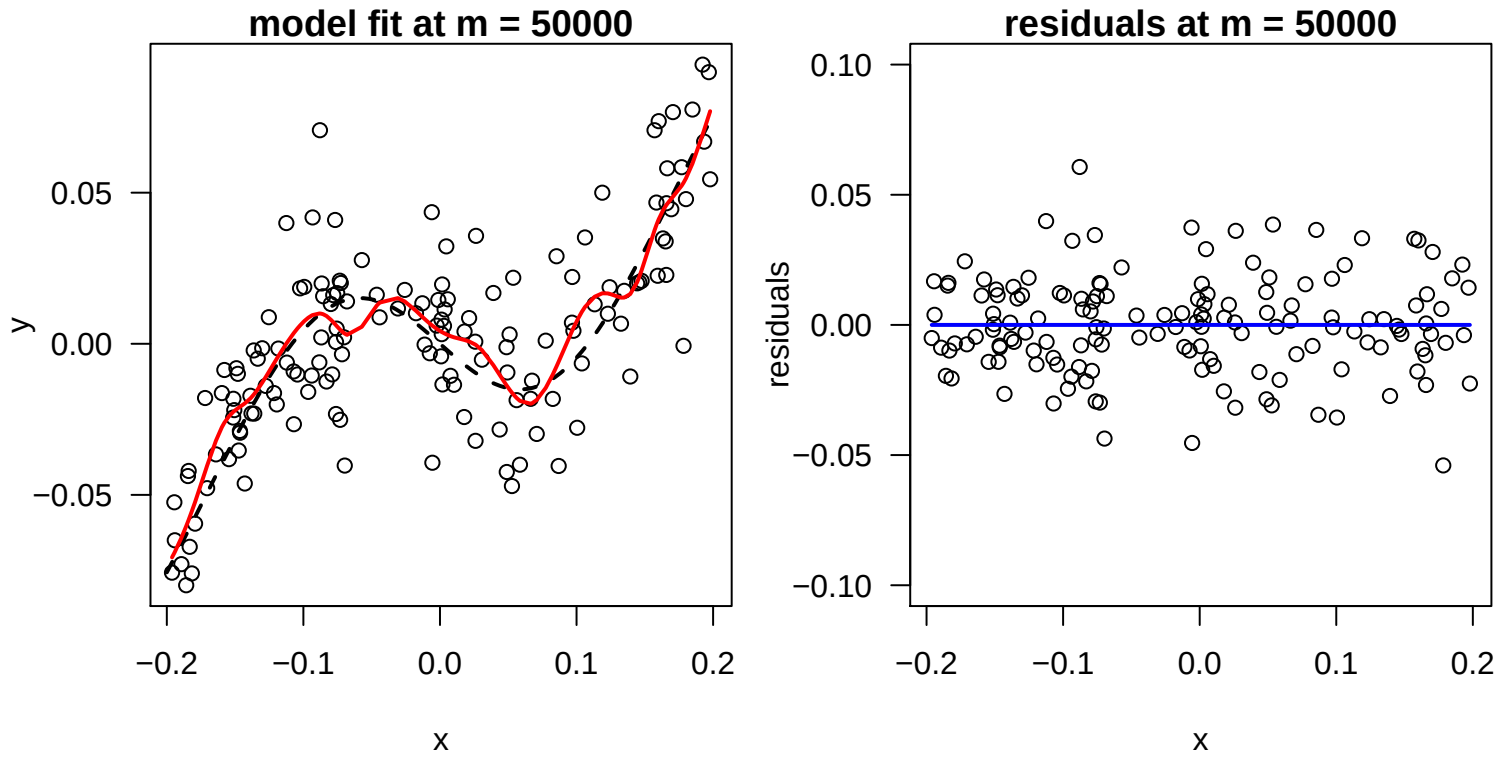
Example for the effect of m_{stop}

Fit an additive model $\mathbb{E}(y_i) = f(x_i)$



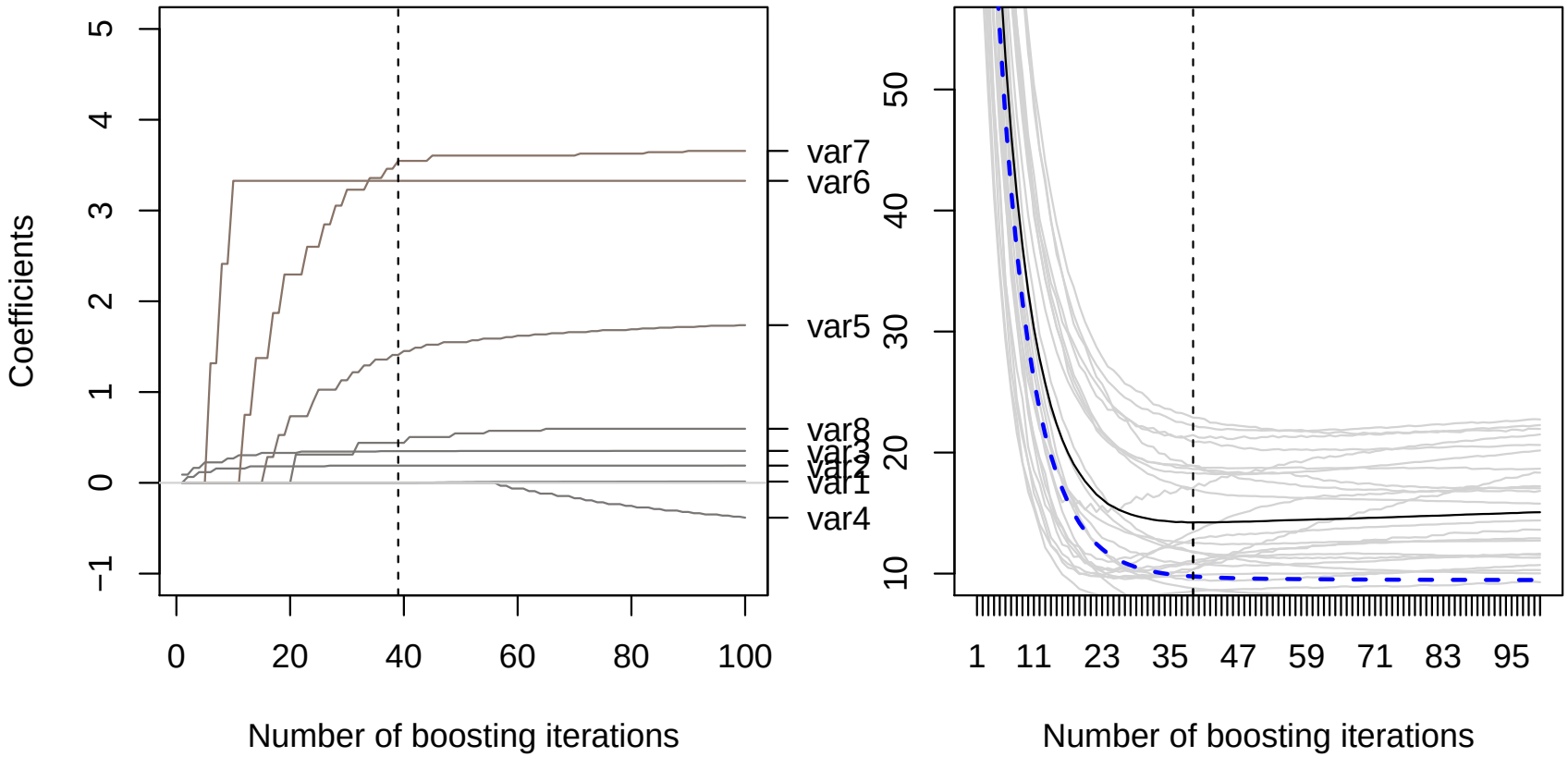
Example for the effect of m_{stop}

Fit an additive model $\mathbb{E}(y_i) = f(x_i)$



Find m_{stop} by 25-fold bootstrap

Estimate LM: $\mathbb{E}(y_i) = \beta_1 \text{var1} + \ldots + \beta_8 \text{var8}$



Left: Coefficient paths against the number of boosting iterations; Right: Out-of-bag risk for each of 25 bootstrap folds; mean oob-risk (black); risk on training data (dashed blue)

Why statistical boosting?

In contrast to Fisher scoring or backfitting, ...

- ... boosting works also many other loss functions (e.g., absolute loss, quantile regression, AUC regression)
- ... boosting is still feasible even if $p \gg n$
- ... boosting allows carrying out variable selection during the fitting process
- ... boosting incorporates shrinkage of effect estimates towards zero (similar to the lasso)
- ... boosting optimizes prediction accuracy
- ... boosting provides a large range of possible types of covariate effects

Boosting can be used to estimate the unknown quantities in almost every statistical regression setting.

R packages for statistical boosting

Gradient boosting

- **mboost** (Hothorn et al.)
- **gamboostLSS** (mboost for GAMLSS, Hofner et al.)
- **gbm** (Ridgeway, G.)

Focus on package **mboost**;

Comment: Boosting as machine learning tool: e.g., R package **xgboost** (Chen et al.)

(2) The R package mboost

mboost: Model-based boosting

The `mboost` package

```
mboost(formula, family = Gaussian(),
       control = boost_control(mstop = 100), ...)
```

- unified boosting framework for various regression settings
- pre-defined base-learners for, e.g., linear, smooth, spatial, random and monotonic effects
- arbitrary combinations of loss-functions and base-learners possible

- Modular nature of boosting.

Case Study: Prediction of Body Fat

Observations of 71 German women (Garcia et al., 2005)

Name	Description
`DEXfat`	body fat measured by DXA (response variable)
`age`	age of the women in years
`waistcirc`	waist circumference
`hipcirc`	hip circumference
`elbowbreadth`	breadth of the elbow
`kneebreadth`	breadth of the knee
`anthro3a`	anthropometric measurement
`anthro3b`	anthropometric measurement
`anthro3c`	anthropometric measurement
`anthro4`	anthropometric measurement

Aim: predict the body fat using the other variables by an interpretable model.

Prerequisites

How to get started?

- Start a recent version of R ($\geq$ 3.2.0).
- Install package **mboost** ($\geq$ 2.3-0):

  ```
  > install.packages("mboost")
  ```

- Load package **mboost**:

  ```
  > library("mboost")
  ```

Prerequisites

How to get started?

- Start a recent version of R ($\geq$ 3.2.0).
- Install package **mboost** ($\geq$ 2.3-0):

  ```
  > install.packages("mboost")
  ```

- Load package **mboost**:

  ```
  > library("mboost")
  ```

Further reading

- This analysis is based on

  B. Hofner, A. Mayr, N. Robinzonov, and M. Schmid (2014). *Model-based boosting in R – A hands-on tutorial using the R package mboost.* Computational Statistics, 29:3–35.

  See there for more details.

Linear Models:

`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., a linear base-learner)

```
glmboost(formula, data = list(),
         center = TRUE, control = boost_control(), ...)
```

Usage in essence very similar to `lm()` and `glm()`.

`formula`: a formula as in `lm()`; e.g.:

```
formula = y ~ x1 + x2 + x3
```

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

`center`: a logical variable indicating if the predictor variables are centered before fitting (default = `TRUE`)
(▶ important for fast convergence of boosting)

`control`: parameters controlling the algorithm (▶ next slide)

Available via "`...`"; possible parameters see `?mboost_fit`:

`family`: used to specify the fitting problem. For example:

```
family = Gaussian()    # default; least squares
family = Binomial()    # logit model
```

[0] Only selected options are displayed throughout this talk!

```
control = boost_control(mstop = 100, nu = 0.1,
                        trace = FALSE)
```

`mstop`: number of *initial* boosting iterations.

`nu`: step length (typically $\in (0, 1]$)

`trace`: should status information be printed during the fitting process?

Specify the regression problem

Possible input for `family`

	continuous response	binary response	count data	ordered response
Mean regression	`Gaussian`			
Median regression	`Laplace`			
Quantile regression	`QuantReg`			
Gamma regression	`GammaReg`			
Logistic regression		`Binomial`		
Poisson regression			`Poisson`	
Negative binomial model			`NBinomial`	
Proportional odds model				`ProppOdds`

Some currently implemented distributions or loss-functions which can be specified via `family` in the fitting functions of **mboost**.

Case Study: Prediction of Body Fat

Observations of 71 German women (Garcia et al., 2005)

Name	Description
`DEXfat`	body fat measured by DXA (response variable)
`age`	age of the women in years
`waistcirc`	waist circumference
`hipcirc`	hip circumference
`elbowbreadth`	breadth of the elbow
`kneebreadth`	breadth of the knee
`anthro3a`	anthropometric measurement
`anthro3b`	anthropometric measurement
`anthro3c`	anthropometric measurement
`anthro4`	anthropometric measurement

Full (Candidate) Model

- Potentially all variables into the model:

```
> glm2 <- glmboost(DEXfat ~ age + waistcirc + hipcirc +
+                  elbowbreadth + kneebreadth + anthro3a +
+                  anthro3b + anthro3c + anthro4,
+                  data = bodyfat)
```

- . . . or use paste to set up the formula:

```
> ## names of predictors
> preds <- names(bodyfat[, names(bodyfat) != "DEXfat"])
> ## build formula
> fm <- as.formula(paste("DEXfat ~",
+                        paste(preds, collapse = "+")))
> fm
```

```
DEXfat ~ age + waistcirc + hipcirc + elbowbreadth + kneebrea
    anthro3a + anthro3b + anthro3c + anthro4
```

```
> glm2 <- glmboost(fm, data = bodyfat)
```

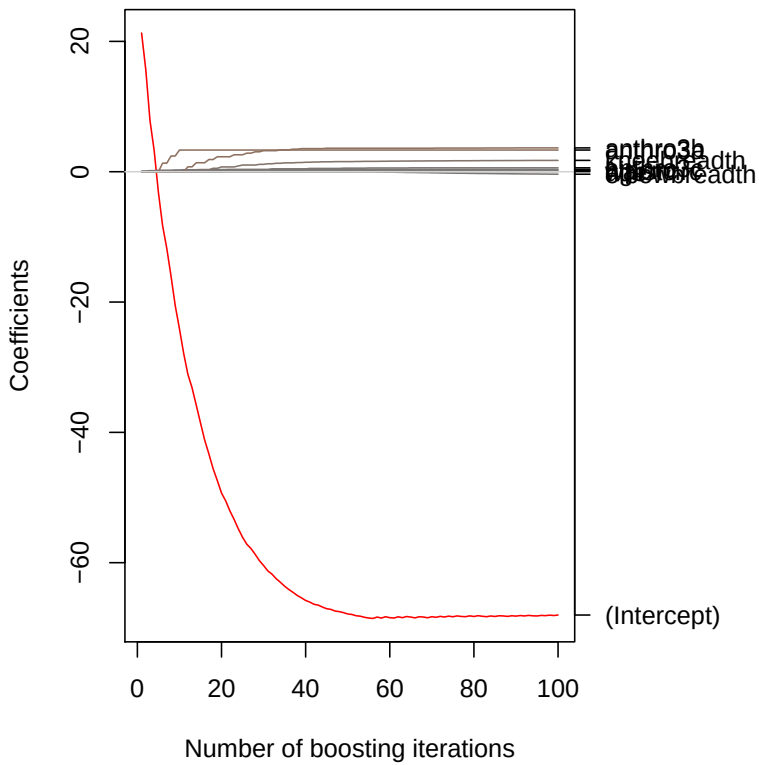
Display Estimated Coefficients

- `which` can be used to define a subset to be displayed:

```
> coef(glm2, which = "age")

      age
0.0136017

> coef(glm2, which = 2)

      age
0.0136017

> coef(glm2, which = "anthro")

 anthro3a  anthro3b  anthro3c   anthro4
3.3268603 3.6565240 0.5953626 0.0000000
```

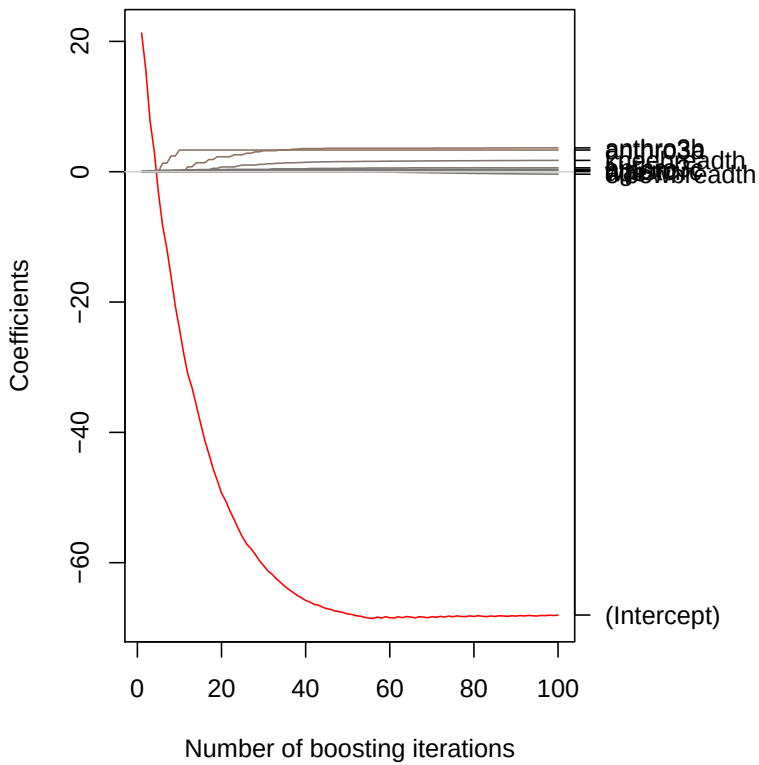
Plot Coefficient Paths

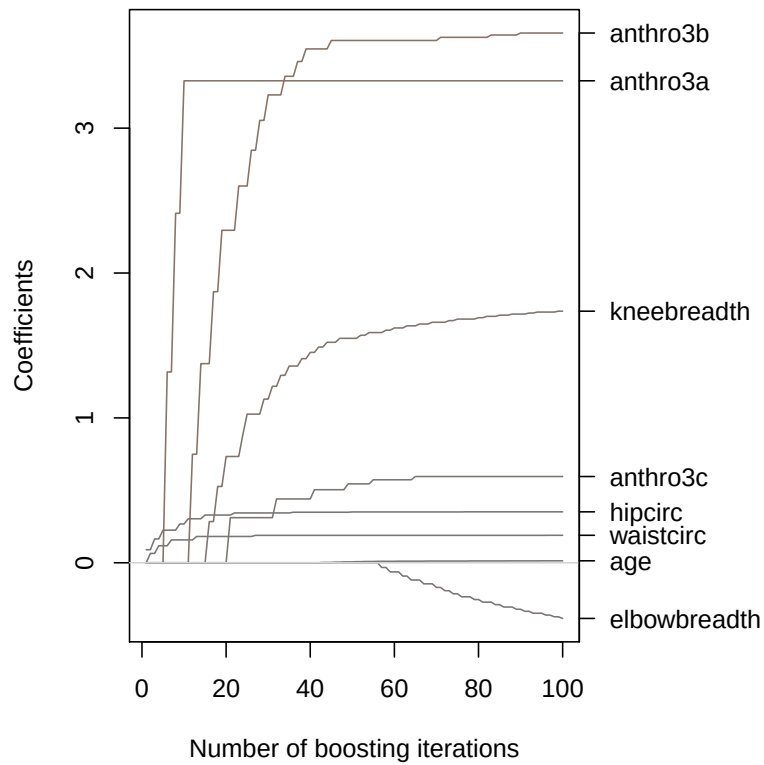
- default plot, offset added to intercept:
  `> plot(glm2, off2int = TRUE)`
- now change `ylim` to the range of the coefficients without intercept (i.e., zoom-in):
  `> plot(glm2, ylim = range(coef(glm2, which = preds)))`



Plot Coefficient Paths

- default plot, offset added to intercept:
  `> plot(glm2, off2int = TRUE)`
- now change `ylim` to the range of the coefficients without intercept (i.e., zoom-in):
  `> plot(glm2, ylim = range(coef(glm2, which = preds)))`





Find the optimal m_{stop}: `cvrisk()` *and Subset Method*

Optimal Stopping Iteration

- Model needs tuning to prevent overfitting
- Determine optimal stopping iteration

- Use cross-validated/bootstrapped estimates of the expected loss to choose an appropriate number of boosting iterations:
  - aims at optimizing prediction 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_{stop}

```
cvrisk(object, folds = cv(model.weights(object)),
       grid = 1:mstop(object),
       papply = mclapply)
```

`object`: the `mboost` model

`folds`: weight matrix that determines the cross-validation samples (▶ see `cv()` on next slide)

`grid`: grid of `mstop` values on which to compute empirical (out-of-bag) risk.

`papply`: use `mclapply` if package **parallel** is available, else run sequentially (▶ see `?cvrisk` for details).

```
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.

`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`

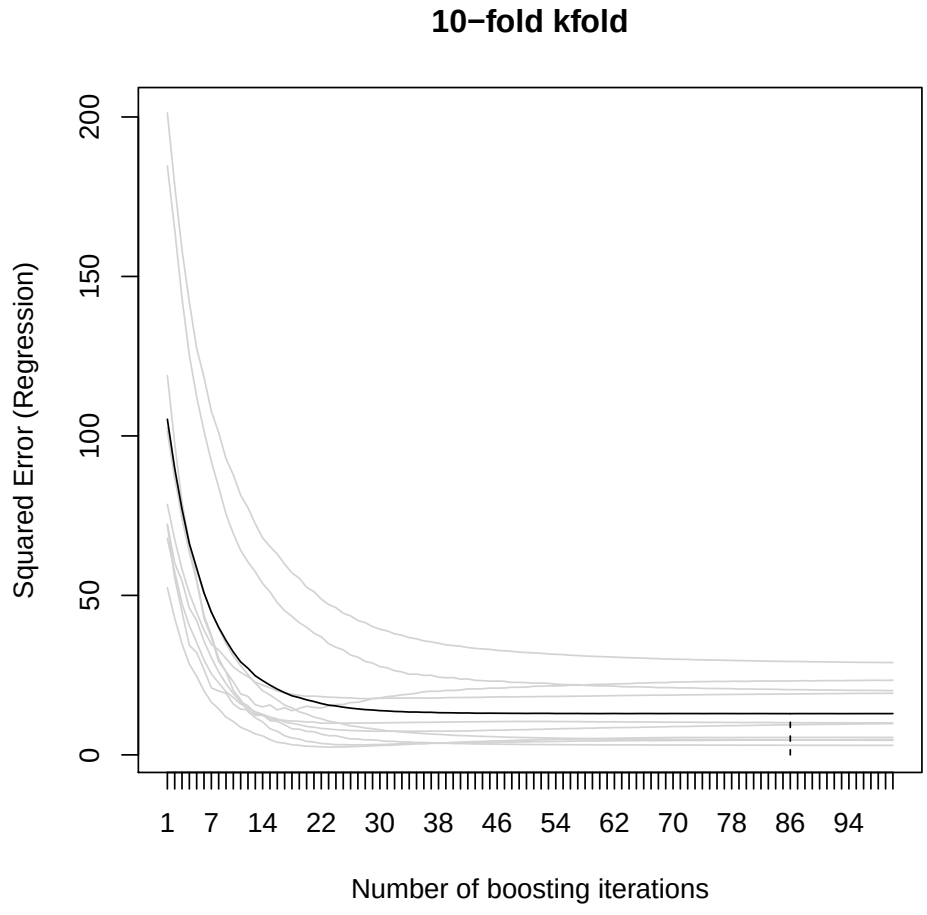
- returns a matrix with `B` columns that determines the weights for each cross-validation run

Case Study: Prediction of Body Fat (ctd.)

Determine m_{stop} by 10-fold cross-validation

```
> set.seed(1907) ## make folds/ analysis reproducible
> cv10f <- cv(model.weights(glm2), type = "kfold")
> cvm <- cvrisk(glm2, folds = cv10f, papply = lapply)
> cvm

         Cross-validated Squared Error (Regression)
         glmboost.formula(formula = DEXfat ~ ., data = bodyfat)

        1         2         3         4         5         6
105.22746  89.82040  77.04107  66.23743  58.54920  50.82022  44.
        9        10        11        12        13        14
 35.88117  32.23113  29.13535  27.14932  24.77990  23.28139  21.

(...)
```

```
> plot(cvm)
> mstop(cvm)

[1] 86
```



Subset

How to change the number of boosting iterations

To increase or reduce the number of boosting steps for the model `glm2`, one can use the indexing / sub-setting operator:

```
glm2[i]   ## set number of boosting iterations to i
glm2 <- glm2[i]   ## the same
```

Attention, non-standard behavior:

- `glm2[i]` *directly changes the model* `glm2`
- no need to save `glm2` under a new name

Alternative way:

```
mstop(glm2) <- i
```

Additive Models:

`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;
`formula = y ~ x1 + x2`
(smooth effects in x1 and x2)

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

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

`control`: parameters controlling the algorithm (▶ Slide 25).

`family`: used to specify the fitting problem.

mboost base-learners

call	effect
`bols(x)`	linear effect: $\boldsymbol{x}^\top \beta$ (with $\boldsymbol{x}^\top = (1, x)$)
`bols(z)`	categorical effect for factor z
`bbs(x)`	smooth effect: $f(x)$ (using P-splines)
`brandom(z)`	random intercept for grouping variable z: b_i

Some currently implemented base-learners of **mboost**.

the function `glmboost()` sets up a model with linear effects;
the function `gamboost()` sets up a model with smooth effects;

Summary

Boosting...

- ... emerged from machine learning.
- ... was later adapted to fit statistical models.
- ... can cope with $p \gg n$.
- ... can carry out variable selection.
- ... has a modular nature (*any* base-learner with *any* loss function).

Limitations:

- Tuning of m_{stop} can lead to relatively long run-times.
- No standard errors for effect estimates, no confidence intervals or p-values.

References

A. L. Garcia, K. Wagner, T. Hothorn, C. Koebnick, H.-J. F. Zunft, and U. Tippo. Improved prediction of body fat by measuring skinfold thickness, circumferences, and bone breadths. *Obesity Research*, 13(3):626–634, 2005.

Benjamin Hofner, Andreas Mayr, Nikolay Robinzonov, and Matthias Schmid. Model-based boosting in R: a hands-on tutorial using the R package mboost. *Computational Statistics*, pages 1–33, 2012. doi: 10.1007/s00180-012-0382-5. URL http://dx.doi.org/10.1007/s00180-012-0382-5.

Torsten Hothorn, Peter Bühlmann, Thomas Kneib, Matthias Schmid, and Benjamin Hofner. *mboost: Model-Based Boosting*, 2016. R package version 2.6-0.

Andreas Mayr, Harald Binder, Olaf Gefeller, and Matthias Schmid. The evolution of boosting algorithms – from machine learning to statistical modeling. *Methods of Information in Medicine*, 2014.