Boosting Functional Regression Models

Hands on Tutorial using FDboost

Sarah Brockhaus David Rügamer

Department of Statistics, LMU Munich

July 24, 2017

References

Brockhaus, S., Ruegamer, D., and Greven, S. (2017). "Boosting Functional Regression Models with FDboost", *ArXiv e-prints* 1705.10662, https://arxiv.org/abs/1705.10662.

Hofner B, Hothorn T, Kneib T, Schmid M (2011). "A framework for unbiased model selection based on boosting", *Journal of Computational and Graphical Statistics*, 20(4), 956971.

Kneib T, Hothorn T, Tutz G (2009). "Variable Selection and Model Choice in Geoadditive Regression Models", *Biometrics*, 65(2), 626634.

Contents

1 Introduction

- In the following you can decide on either fitting

  (2.1) different scalar-on-function regression models (see page 2) or

  (2.2) different function-on-function regression models (see page 3)

  using the `fuelSubset` or the `emotion` data set, respectively. Make yourself familiar with the according data set.

- For each model fit in exercise (2.1) or (2.2), go through the following steps

  1. **Model fit**:
     Fit the model `mod` using the function `FDboost`. Use the argument `control = boost_control(mstop = ...)` to change the number of iterations for the boosting algorithm. Make sure to facilitate a fair base-learner selection (see section 3 for more details).
  2. **Empirical risk**:
     In order to estimate the empirical risk of the model in each iteration, use
     - ▷ `er <- cvrisk(mod)` for (2.1),
     - ▷ `er <- applyFolds(mod)` for (2.2).
  3. **Optimal iteration**:
     Determine the optimal stopping iteration `ms` by using `mstop(er)` and visualize the results in `er` using `plot(er)`.
  4. **Update model**:
     Update the fitted model using the optimal stopping iteration: `mod[ms]`.
  5. **Model results**:
     Check the model summary using `summary(mod)`, e.g., look at the selection probability for each baselearner.
  6. **Visualize results**:
     Plot the coefficients using `plot(mod)` and interpret the results.
  7. **Extras**:
     Have a look at
     - ▷ the coefficients using `str(coef(mod), 1)`,
     - ▷ the design matrix `str(extract(mod, "design"), 1)` or
     - ▷ the degrees of freedom `extract(mod, "df")`.

2 Exercises

2.1 Scalar-on-function regression

Use the `fuelSubset` data to fit different scalar-on-function regression models in the following (i.e., set `data = fuelSubset` in the `FDboost`-call). In each case, use the variable `heatan` as a scalar response. For scalar response regression, the argument `timeformula = NULL`.

1. Fit an additive model with intercept using `1` in the formula argument and a non-linear effect for `h2o` using the `bbs(...)`-base-learner.

2. Extend the model by two linear functional effects for `UVVIS` and `NIR` using the base-learner `bsignal(...)`. First, center the functional covariates to center their effects around zero.

   ```
   fuelSubset$UVVIS <- scale(fuelSubset$UVVIS, scale = FALSE)
   fuelSubset$NIR <- scale(fuelSubset$NIR, scale = FALSE)
   ```

   For the argument `s` in `bsignal` use the observed time grid for the respective covariate (`uvvis.lambda`, `nir.lambda`).

3. Fit another model, in which the `bsignal` base-learner is replaced by the functional principal components base-learner `bfpc`.

4. Use `predict` to compute predictions for the previous two models and compare the predictions.

5. Advanced Exercise: Compute bootstrap intervals for the estimated coefficients using the `bootstrapCI` function and plot the intervals using the `plot` method.

2.2 Function-on-function regression

Use the `emotion` data to fit different function-on-function regression models in the following (i.e., set `data = emotion` in the `FDboost`-call). In each case, use the variable `EMG` as a functional response. For functional response regression, use the argument `timeformula = ~ bbs(t, df = 3)`.

1. Fit a pure intercept model.
2. Fit a function-on-scalar model with a subject effect and an effect for `power` using the base-learner `bolsc(...)` in both cases.
3. In order to reduce computational burden for the following model fits, create a subset of the `emotion` data as follows and make yourself familiar with the data structure.

```r
# define subset for a certain game condition
subset <- emotion$control == "high" &
  emotion$game_outcome == "gain" &
  emotion$power == "low"
emotionHGL <- list()

# subset scalar variables
emotionHGL$subject <- emotion$subject[subset]

# subset functional variables
emotionHGL$EMG <- emotion$EMG[subset,]
emotionHGL$EEG <- emotion$EEG[subset,]
# center the functional covariate per time-point
emotionHGL$EEG <- scale(emotionHGL$EEG, scale = FALSE)

# keep the evaluation points
emotionHGL$s <- emotionHGL$t <- emotion$t
```

4. Use the `emotionHGL` subset to fit a function-on-function regression model with a signal effect for `EEG` using the base-learner `bsignal(EEG, s = s)`.
5. Replace the signal effect with a concurrent effect using the base-learner `bconcurrent(EEG, s = s, time = t)`.
6. Replace the concurrent effect with a historical effect using the base-learner `bhist(EEG, s = s, time = t, limits = function(s,t) s <= t)`.

3 Additional information: Fair base-learner selection

Excerpts from Brockhaus et al. (2017):

> *To ensure a fair selection of base-learners within the course of all iterations, it is important to specify equal degrees of freedom for each base-learner. If base-learners exhibit different degrees of freedom, selection is biased towards more flexible base-learners with higher degrees of freedom as they are more likely to yield larger improvements of the fit in each iteration (see Hofner et al., 2011 for more details). It is recommended to use a rather small number of degrees of freedom for all base-learners to work with weak learners (Kneib et al. 2009; Hofner et al. 2011).*
>
> *For a fair selection of base-learner, additional care is needed for functional responses as only some of the base-learners in the formula are expanded by the base-learner in timeformula. In particular, all base-learners listed in Table 2 are expanded by timeformula, whereas base-learners given in Table 3 are not expanded by the timeformula. For the row-wise tensor product and the Kronecker product of two base-learners, the degrees of freedom for the combined base-learner is computed as product of the two marginally specified degrees of freedom. For instance,* `formula = y ~ bbsc(z, df = 3) + bhist(x, s = s, df = 12)` *and* `timeformula = ~ bbs(t, df = 4)` *implies $3 \cdot 4 = 12$ degrees of freedom for the first combined base-learner and 12 degrees of freedom for the second base-learner. The call* `extract(object, "df")` *displays the degrees of freedom for each base-learner in an FDboost object.*

4 Additional information: Possible covariate effects and base-learners

Following tables are taken from Brockhaus et al. (2017).

covariate(s)	type of effect	$h_j(x, t)$
(none)	smooth intercept	$\beta_0(t)$
scalar covariate z	linear effect	$z\beta(t)$
	smooth effect	$f(z, t)$
two scalars z_1, z_2	linear interaction	$z_1 z_2 \beta(t)$
	functional varying coefficient	$z_1 f(z_2, t)$
	smooth interaction	$f(z_1, z_2, t)$
functional covariate $x(s)$	linear functional effect	$\int_{\mathcal{S}} x(s)\beta(s, t)\, ds$
scalar z and functional $x(s)$	linear interaction	$z \int_{\mathcal{S}} x(s)\beta(s, t)\, ds$
	smooth interaction	$\int_{\mathcal{S}} x(s)\beta(z, s, t)\, ds$
functional covariate $x(s)$,	concurrent effect	$x(t)\beta(t)$
with $\mathcal{S} = \mathcal{T} = [T_1, T_2]$	historical effect	$\int_{T_1}^{t} x(s)\beta(s, t)\, ds$
	lag effect, with lag $\delta > 0$	$\int_{t-\delta}^{t} x(s)\beta(s, t)\, ds$
	lead effect, with lead $\delta > 0$	$\int_{T_1}^{t-\delta} x(s)\beta(s, t)\, ds$
	effect with t-specific integration limits $[l(t), u(t)]$	$\int_{l(t)}^{u(t)} x(s)\beta(s, t)\, ds$
grouping variable g	group-specific smooth intercepts	$\beta_g(t)$
grouping variable g and scalar z	group-specific linear effects	$z\beta_g(t)$
curve indicator i	curve-specific smooth residuals	$e_i(t)$

Table 1: Overview of some possible covariate effects that can be represented within the framework of functional regression.

additive predictor $h(\boldsymbol{x},t) = \sum_j h_j(\boldsymbol{x},t)$	call
$\beta_0(t)$	`y ~ 1`
$\beta_0(t) + z_1\beta_1(t)$	`y ~ 1 + bolsc(z1)`
$\beta_0(t) + f_1(z_1,t)$	`y ~ 1 + bbsc(z1)`
$\beta_0(t) + z_1\beta_1(t) + z_2\beta_2(t) + z_1z_2\beta_3(t)$	`y ~ 1 + bolsc(z1) + bolsc(z2) + bols(z1) %Xc% bols(z2)`
$\beta_0(t) + z_1\beta_1(t) + f_2(z_2,t) + z_1 f_3(z_2,t)$	`y ~ 1 + bolsc(z1) + bbsc(z2) + bols(z1) %Xc% bbs(z2)`
$\beta_0(t) + f_1(z_1,t) + f_2(z_2,t) + f_3(z_1,z_2,t)$	`y ~ 1 + bbsc(z1) + bbsc(z2) + bbs(z1) %Xc% bbs(z2)`
$\beta_0(t) + \int_{\mathcal{S}} x(s)\beta_1(s,t)\,ds$	`y ~ 1 + bsignal(x, s = s)`
	`y ~ 1 + bfpc(x, s = s)`
$\beta_0(t) + z\beta_1(t) + \int_{\mathcal{S}} x(s)\beta_2(s,t)\,ds + z\int_{\mathcal{S}} x(s)\beta_3(s,t)\,ds$	`y ~ 1 + bolsc(z) + bsignal(x, s = s) + bsignal(x, s = s) %X% bolsc(z)`

Table 2: Additive predictors that can be represented within the array framework.

additive predictor $h(x,t) = \sum_j h_j(x,t)$	call
$\beta_0(t) + x(t)\beta(t)$	`y ~ 1 + bconcurrent(x, s = s, time = t)`
$\beta_0(t) + \int_{T_1}^{t} x(s)\beta(s,t)\,ds$	`y ~ 1 + bhist(x, s = s, time = t)`
$\beta_0(t) + \int_{t-\delta}^{t} x(s)\beta(s,t)\,ds$	`y ~ 1 + bhist(x, s = s, time = t,` `limits = limitsLag)`*
$\beta_0(t) + \int_{T_1}^{t-\delta} x(s)\beta(s,t)\,ds$	`y ~ 1 + bhist(x, s = s, time = t,` `limits = limitsLead)`*
$\int_{l(t)}^{u(t)} x(s)\beta(s,t)\,ds$	`y ~ 1 + bhist(x, s = s, time = t, limits = mylimits)`
$\beta_0(t) + z\beta_1(t) + \int_{T_1}^{t} x(s)\beta_2(s,t)\,ds$ $+ z\int_{T_1}^{t} x(s)\beta_3(s,t)\,ds$	`y ~ 1 + bolsc(z) + bhist(x, s = s, time = t)` `+ bhistx(x) %X% bolsc(z)`

Table 3: Additive predictors that contain effects that cannot be separated into an effect in covariate direction and an effect in t direction. These effects in `formula` are not expanded by the `timeformula`. We give examples for general limit functions `mylimits` in this section. In `bhistx()`, the variable `x` has to be of class `hmatrix`, please see the manual of `bhistx()` for details.