universität
innsbruck

Scaleable Distributional Learning

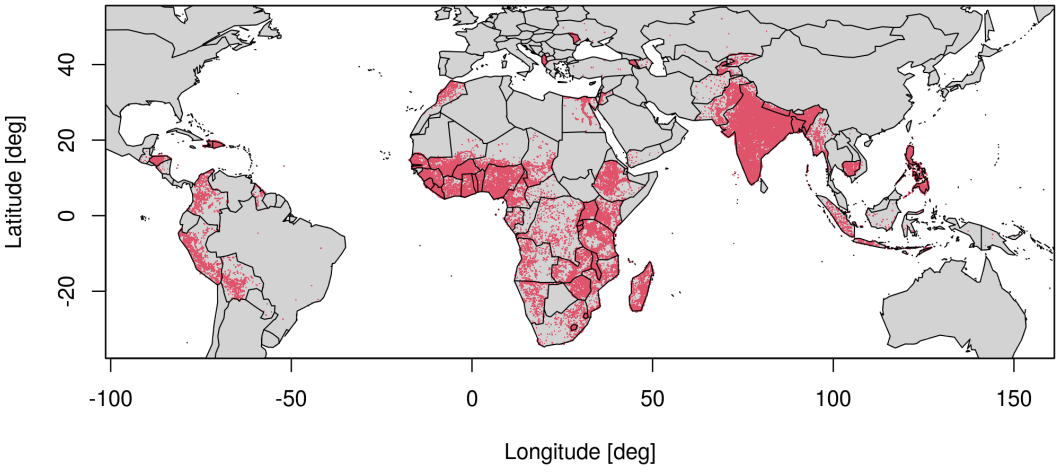
Nikolaus Umlauf
http://nikum.org/

Childhood malnutrition

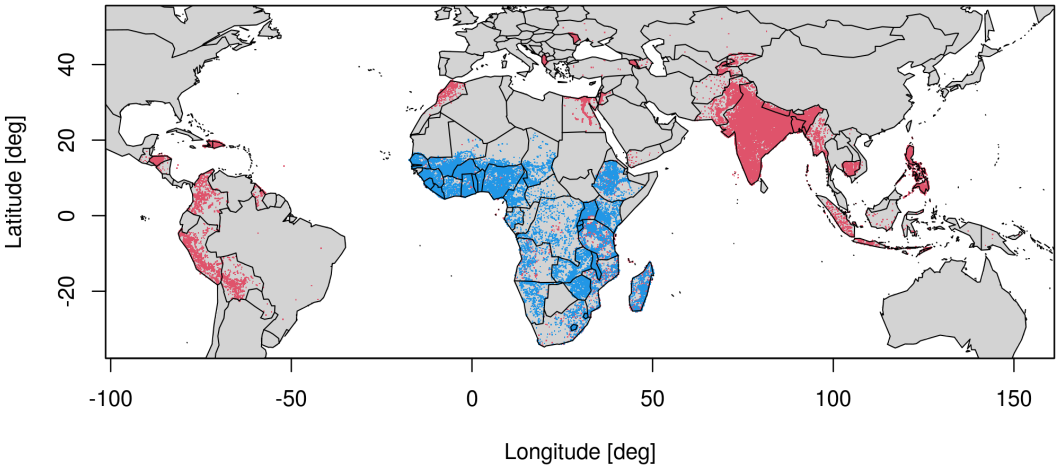
Joint work with:
Johannes Seiler, Mattias Wetcher, Thorsten Simon & Stefan Lang.

- Project aiming to better explain the problems of childhood malnutrition in low- and middle-income countries.
- Contribute to monitoring of the Sustainable Development Goals (SGD).
- We compiled a brand new data set using DHS data.
- Data on global conflicts, topography and environmental data from satellite earth observations (NDVI), temperature and precipitation data from ERA5.
- Data from 1990–2019 with $n > 3M$ observations.

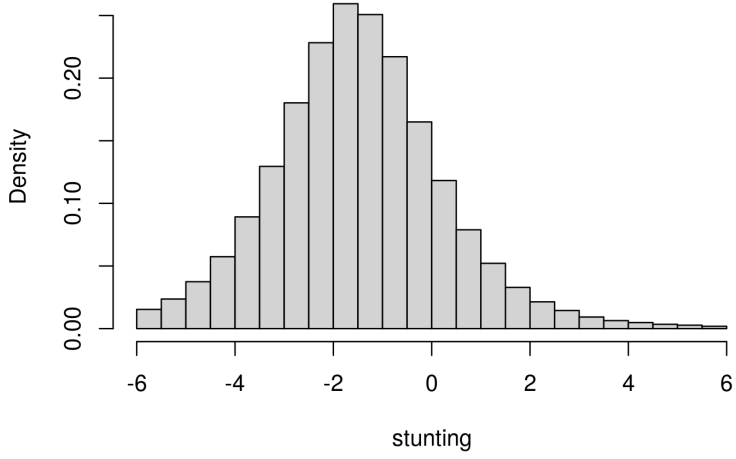
Childhood malnutrition



Childhood malnutrition



Childhood malnutrition



Computational challenges

- Distributional regression using (very) large data sets?
- Variable selection?
- Inference?

Model specification

Any parameter of a population distribution $\mathcal{D}$ may be modeled by explanatory variables

$$y \sim \mathcal{D}\left(\theta_1(\mathbf{x};\boldsymbol{\beta}_1),\ \ldots,\ \theta_K(\mathbf{x};\boldsymbol{\beta}_K)\right),$$

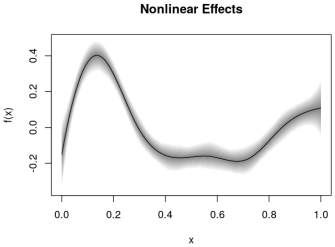
with $\boldsymbol{\beta} = (\boldsymbol{\beta}_1^\top, \ldots, \boldsymbol{\beta}_K^\top)^\top$.

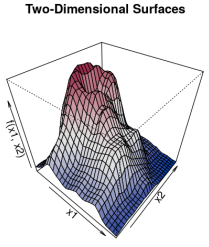
Each parameter is linked to a structured additive predictor

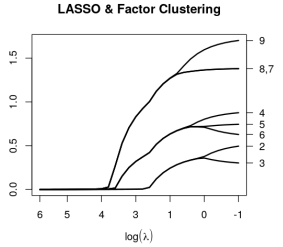
$$h_k(\theta_k(\mathbf{x};\boldsymbol{\beta}_k)) = f_{1k}(\mathbf{x};\boldsymbol{\beta}_{1k}) + \ldots + f_{J_k k}(\mathbf{x};\boldsymbol{\beta}_{J_k k}),$$

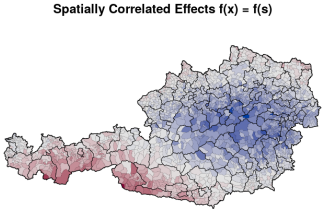
- $j = 1, \ldots, J_k$ and $k = 1, \ldots, K$.
- $h_k(\cdot)$: Link functions for each distribution parameter.
- $f_{jk}(\cdot)$: Model terms of one or more variables.

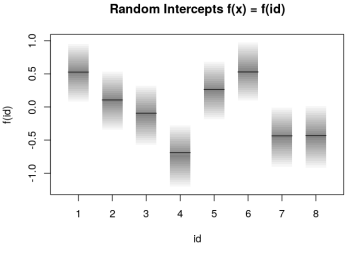
Model terms $f_{jk}(\cdot)$

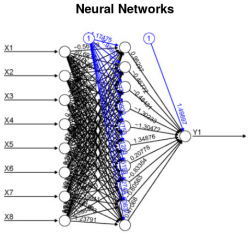












Estimation

The main building block of regression model algorithms is the probability density function $d_y(\mathbf{y}|\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K)$.

Estimation typically requires to evaluate the log-likelihood

$$\ell(\boldsymbol{\beta}; \mathbf{y}, \mathbf{X}) \;\; = \;\; \sum_{i=1}^{n} \log \; d_y(y_i; \theta_1(\mathbf{x}_i; \boldsymbol{\beta}_1), \; \ldots \; , \; \theta_K(\mathbf{x}_i; \boldsymbol{\beta}_K)),$$

with $\mathbf{X} = (\mathbf{X}_1, \ldots, \mathbf{X}_K)$.

The log-posterior (frequentist penalized log-likelihood)

$$\log \, \pi(\boldsymbol{\beta}, \boldsymbol{\tau}; \mathbf{y}, \mathbf{X}, \boldsymbol{\alpha}) \propto \ell(\boldsymbol{\beta}; \mathbf{y}, \mathbf{X}) + \sum_{k=1}^{K} \sum_{j=1}^{J_k} \left[\log \, p_{jk}(\boldsymbol{\beta}_{jk}; \boldsymbol{\tau}_{jk}, \boldsymbol{\alpha}_{jk}) \right],$$

where $p_{jk}(\cdot)$ are priors, $\boldsymbol{\tau}_{jk}$ (smoothing) variances and $\boldsymbol{\alpha}_{jk}$ fixed hyper parameters.

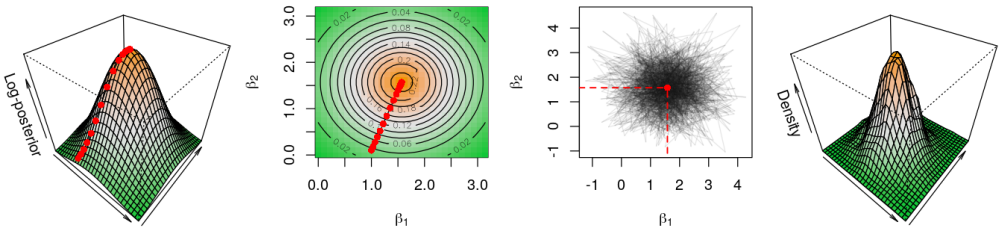
Estimation

Bayesian point estimates of parameters are obtained by:

1. Maximization of the log-posterior for posterior mode estimation.
2. Solving high dimensional integrals, e.g., for posterior mean or median estimation.

Problems 1 and 2 are commonly solved by computer intensive iterative algorithms of the following type:

$$(\boldsymbol{\beta}^{[t+1]}, \boldsymbol{\tau}^{[t+1]}) = \mathtt{U}(\boldsymbol{\beta}^{[t]}, \boldsymbol{\tau}^{[t]}; \mathbf{y}, \mathbf{X}, \boldsymbol{\alpha}).$$



Scaleable distributional learning

Consider the following updating scheme

$$\boldsymbol{\beta}_k^{[t+1]} = \mathtt{U}_k(\boldsymbol{\beta}_k^{[t]}; \cdot) = \boldsymbol{\beta}_k^{[t]} - \mathbf{H}_{kk}\left(\boldsymbol{\beta}_k^{[t]}\right)^{-1}\mathbf{s}\left(\boldsymbol{\beta}_k^{[t]}\right).$$

Assuming model terms that can be written as a matrix product of a design matrix and coefficients we obtain an iteratively weighted least squares scheme given by

$$\boldsymbol{\beta}_{jk}^{[t+1]} = \mathtt{U}_{jk}(\boldsymbol{\beta}_{jk}^{[t]}; \cdot) = (\mathbf{X}_{jk}^\top \mathbf{W}_{kk}\mathbf{X}_{jk} + \mathbf{G}_{jk}(\boldsymbol{\tau}_{jk}))^{-1}\mathbf{X}_{jk}^\top \mathbf{W}_{kk}(\mathbf{z}_k - \boldsymbol{\eta}_{k,-j}^{[t+1]}),$$

with working observations $\mathbf{z}_k = \boldsymbol{\eta}_k^{[t]} + \mathbf{W}_{kk}^{-1\,[t]}\mathbf{u}_k^{[t]}$, working weights $\mathbf{W}_{kk}^{-1\,[t]}$ and score vector $\mathbf{u}_k^{[t]}$.

Scaleable distributional learning

Instead of using all observations of the data, we only use a randomly chosen **subset** denoted by the subindex [**s**] in one updating step

$$\begin{aligned}\boldsymbol{\beta}_{jk}^{[t+1]} &= \nu \cdot (\mathbf{X}_{[\mathbf{s}],jk}^{\top}\mathbf{W}_{[\mathbf{s}],kk}\mathbf{X}_{[\mathbf{s}],jk} + \mathbf{G}_{jk}(\boldsymbol{\tau}_{jk}))^{-1}\mathbf{X}_{[\mathbf{s}],jk}^{\top}\mathbf{W}_{[\mathbf{s}],kk}(\mathbf{z}_{[\mathbf{s}],k} - \boldsymbol{\eta}_{[\mathbf{s}],k,-j}^{[t+1]}) + \\ &\quad (1-\nu)\cdot\boldsymbol{\beta}_{jk}^{[t]},\end{aligned}$$

where ν is a weight parameter which specifies how much the parameters at iteration $t+1$ are influenced by parameters of the previous iteration t.

Use **flat file** format for each $\mathbf{X}_{jk}$, i.e., only batch $[s]$ is in memory. This way, we can estimate models with **really** large datasets!

Scaleable distributional learning

Mimics a second order **stochastic gradient descent** (SGD) algorithm

$$\boldsymbol{\beta}_{jk}^{[t+1]} = \boldsymbol{\beta}_{jk}^{[t]} + \nu \cdot (\boldsymbol{\beta}_{jk,[\mathbf{s}]} - \boldsymbol{\beta}_{jk}^{[t]}) = \boldsymbol{\beta}_{jk}^{[t]} + \nu \cdot \boldsymbol{\delta}_{jk}^{[t]},$$

and $\boldsymbol{\delta}_{jk}^{[t]}$ is composed from first and second order derivative information with

$$\begin{aligned}
\boldsymbol{\delta}_{jk}^{[t]} &= \boldsymbol{\beta}_{jk,[\mathbf{s}]} - \boldsymbol{\beta}_{jk}^{[t]} \\
&= \left[\boldsymbol{\beta}_{jk}^{[t]} - \mathbf{H}_{[\mathbf{s}],kk} \left(\boldsymbol{\beta}_{jk}^{[t]}\right)^{-1} \mathbf{s}_{[\mathbf{s}]} \left(\boldsymbol{\beta}_{jk}^{[t]}\right)\right] - \boldsymbol{\beta}_{jk}^{[t]} \\
&= -\mathbf{H}_{[\mathbf{s}],kk} \left(\boldsymbol{\beta}_{jk}^{[t]}\right)^{-1} \mathbf{s}_{[\mathbf{s}]} \left(\boldsymbol{\beta}_{jk}^{[t]}\right)
\end{aligned}$$

Hence, the updating step length is adaptive.

Scaleable distributional learning

The idea is to select $\boldsymbol{\tau}_{jk}$ using a stepwise algorithm which is based on an **"out-of-sample" criterion**, i.e., the criterion $C(\cdot)$ is evaluated on another batch denoted by $[\tilde{\mathbf{s}}]$, $C_{[\tilde{\mathbf{s}}]}(\cdot)$ respectively, i.e.

$$\tau_{ljk}^{[t+1]} \leftarrow \underset{\tau_{ljk}^{\star} \in \mathcal{I}_{ljk}}{\arg\min}\ C_{[\tilde{\mathbf{s}}]}(U_{jk}(\boldsymbol{\beta}_{jk}^{[t]}, \tau_{ljk}^{\star}; \cdot)),$$

where $\mathcal{I}_{ljk}$ is a search interval for $\tau_{ljk}^{[t+1]}$, e.g.,

$$\mathcal{I}_{ljk} = [\tau_{ljk}^{[t]} \cdot 10^{-1}, \tau_{ljk}^{[t]} \cdot 10].$$

Scaleable distributional learning

Some interesting features:

1. Set, e.g., $\nu = 0.1$, convergence after visiting m batches [**s**].
2. Only update if **"out-of-sample" log-likelihood** is **increased**.
3. **Boosting** for **variable selection**: Update only $f_{jk}(\cdot)$ with greatest contribution in "out-of-sample" log-likelihood.
4. **Bagging**: If $\nu = 1$, each update is so to say a **"sample"**. Convergence similar to MCMC algorithms, i.e., if $\beta_{jk}^{[t+1]}$ start fluctuating around a certain level.
5. **Slice sample** τ_{ljk} under $C_{[\tilde{\mathbf{s}}]}(\cdot)$, **much faster**!

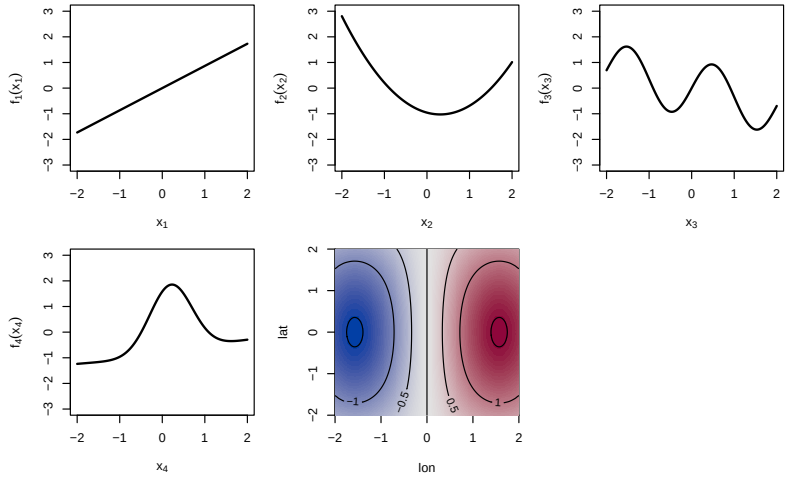
Simulation study

- We consider models using the following predictors:

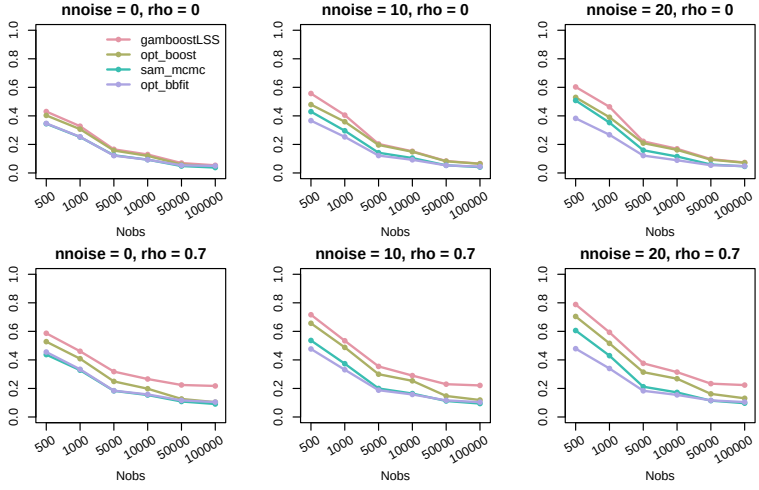
$$\begin{aligned} \eta_\mu &= f_1(x_1) + f_3(x_3) + f_{2d}(\text{lon}, \text{lat}) \\ \eta_\sigma &= f_2(x_2) + f_3(x_3) + f_4(x_4) \text{ with } x_j, \text{lon}, \text{lat} \sim U(-2, 2). \end{aligned}$$

- We investigate performance for `NO()`, `GA()` and `ZAP()` distribution,
- using $n = 500; 1000; 5000; 10000; 50000; 100000$ observations.
- Moreover, we study performance using nnoise $= 0, 10, 20$ variables.
- Two settings for covariate data: uncorrelated and pairwise correlation of $\rho = 0.7$.
- 100 replications each, evaluation using bias, mse, (continous-) ranked probability score ((C)RPS).
- Algorithms: *gamboostLSS*, `sam_MCMC`, `opt_boost`, `opt_bbfit`.

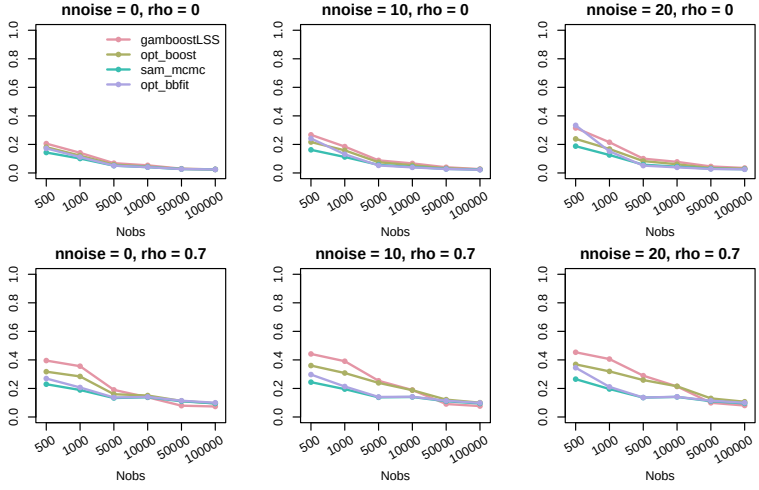
Simulation study



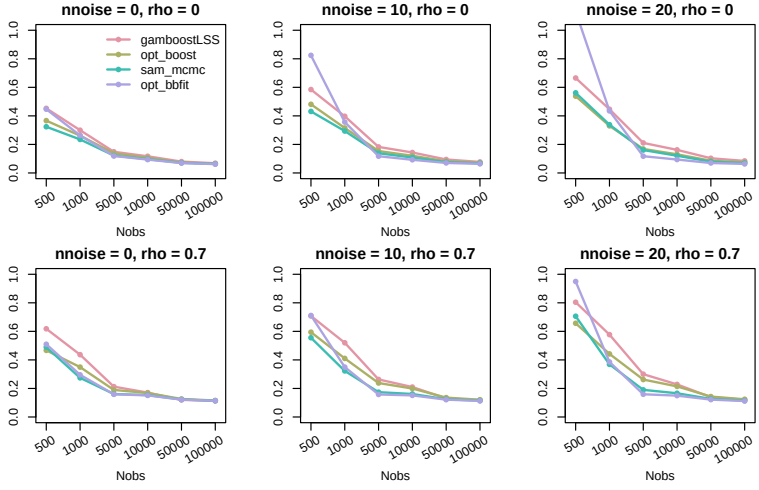
Simulation study: `NO()`, $\sqrt{\text{bias}}$ η_μ



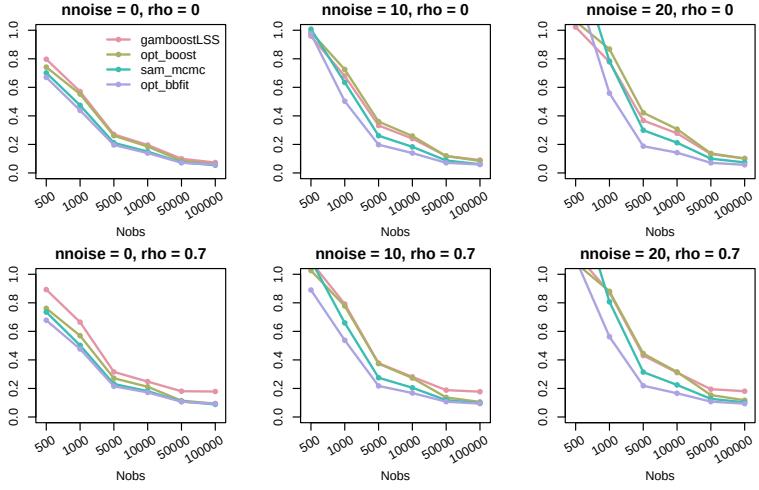
Simulation study: `GA()`, $\sqrt{\text{bias}}$ η_μ



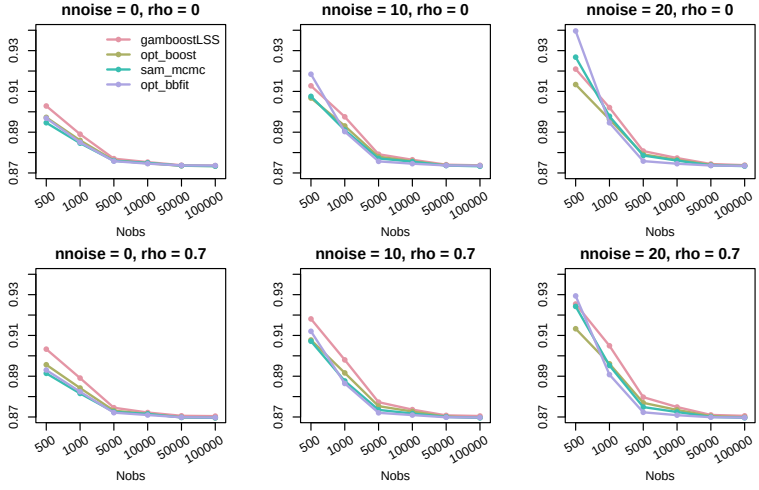
Simulation study: `GA()`, $\sqrt{\text{bias}}$ η_σ



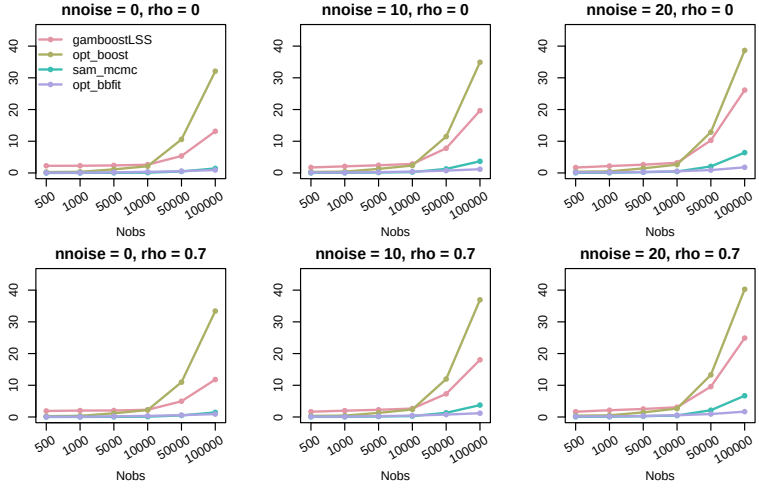
Simulation study: `ZAP()`, $\sqrt{\text{bias}}$ η_σ



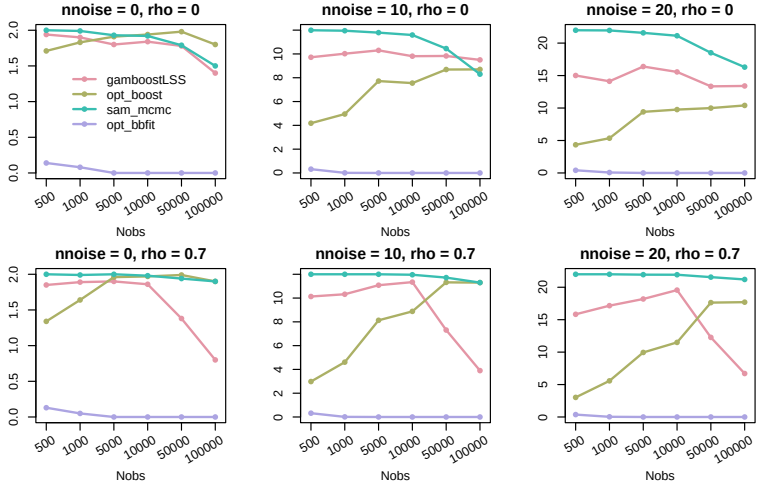
Simulation study: `GA()`, $\sqrt{\text{CRPS}}$



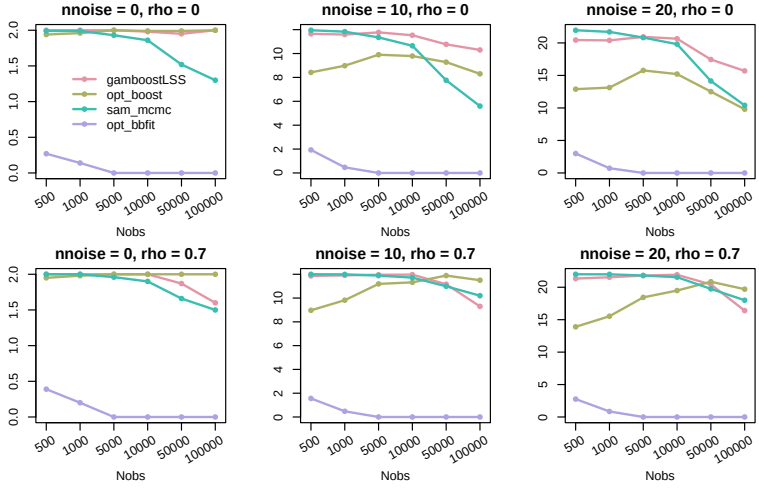
Simulation study: `NO()`, `GA()`, `ZAP()`; runtime [h]



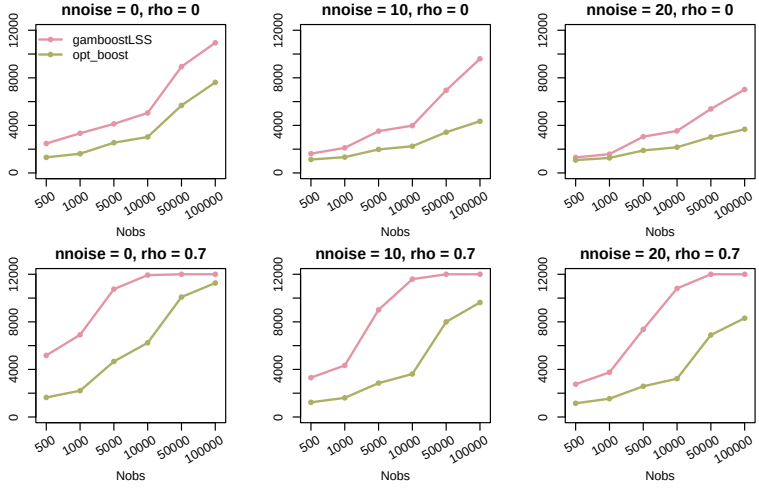
Simulation study: `NO()`, FP μ



Simulation study: `NO()`, FP σ



Simulation study: `NO()`; mstop



Application

Example: Search distribution.

Define the batchsize.

```
R> bs <- 2000
```

Generate batches.

```
R> batch_ids <- lapply(1:200, function(...) {
+    sample(1:nrow(d), size = bs, replace = FALSE)
+  })
```

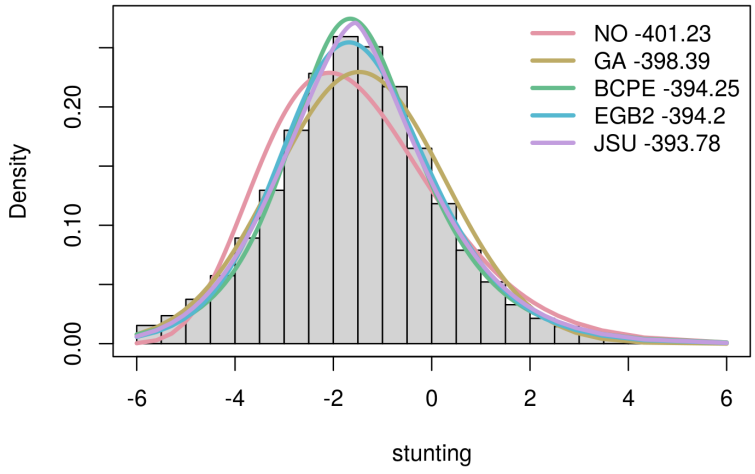
Estimate model.

```
R> b <- bamlss(y ~ 1, data = d, family = JSU,
+    sampler = FALSE, optimizer = opt_bbfitp, slice = 10, aic = TRUE, K = 2,
+    batch_ids = batch_ids)
```

Compute log-likelihood.

```
R> logLik(b, newdata = nd)
```

Application



Application

Example: Boosting flavour with *ff* data frame.

Set up a model formula.

```
R> f <- list(
+    stunting ~ s(cage) + s(bord) + s(hhs) + s(x, y) + ...,
+             ~ s(cage) + s(bord) + s(hhs) + s(x, y) + ...
+  )
```

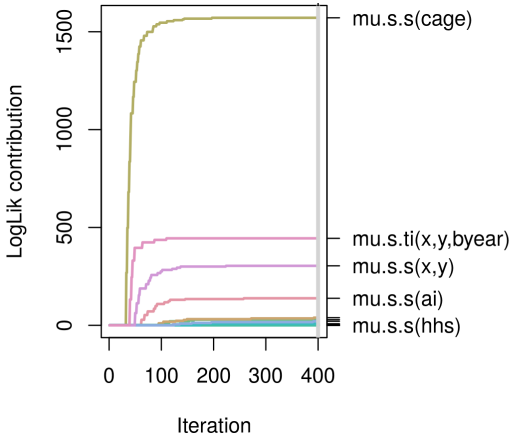
Estimate model.

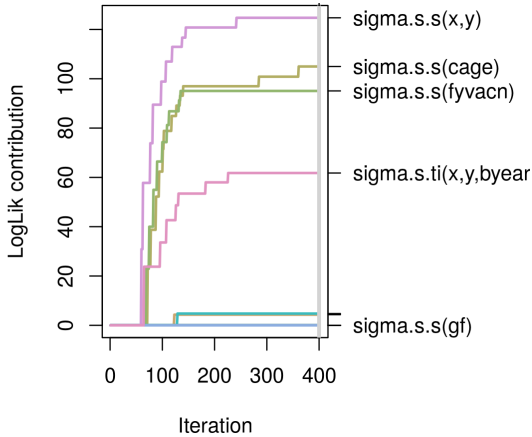
```
R> b <- bamlss(f, data = dff, family = JSU,
+    sampler = FALSE, optimizer = opt_bbfit,
+    batch_ids = batch_ids, select = TRUE, aic = TRUE, always = FALSE,
+    eps_loglik = 0.001, K = 2, overwrite = TRUE, delete = FALSE,
+    ff_name = ff_name)
```

Plot results.

```
R> contribplot(b)
```

Application





Application

Example: Bagging type flavour with slice sampling.

Extract formula.

```
R> nf <- new_formula(b)
```

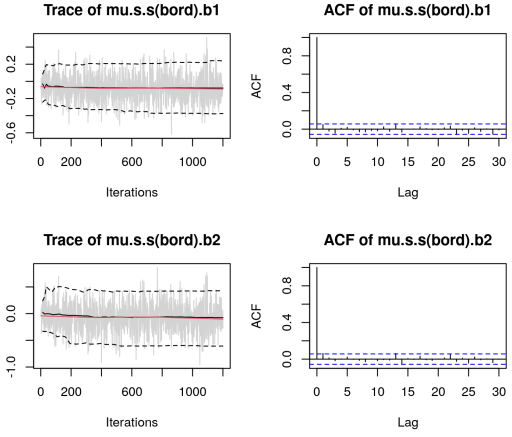
Estimate model using ff processed data.

```
R> m <- bamlss(nf, data = dff, family = JSU,
+    sampler = FALSE, optimizer = opt_bbfitp,
+    batch_ids = batch_ids, aic = TRUE, slice = TRUE,
+    ff_name = ff_name)
```

Afterwards, all extractor functions provided by *bamlss* can be used, e.g., `summary()`, `predict()`, `plot()`, etc.

Application

```
R> plot(m, which = "samples")
```



References

- Umlauf, Klein, and Zeileis (2018). *BAMLSS: Bayesian Additive Models for Location, Scale and Shape (and Beyond)*. Journal of Computational and Graphical Statistics, doi:10.1080/10618600.2017.1407325.
- Groll, Hambucker, Kneib, and Umlauf (2019). *LASSO-Type Penalization in the Framework of Generalized Additive Models for Location Scale and Shape*. Computational Statistics & Data Analysis, doi:10.1016/j.csda.2019.06.005.
- Umlauf et al. (2021). *bamlss: Bayesian Additive Models for Location Scale and Shape (and Beyond)*. R package version 1.1-6, http://CRAN.R-project.org/package=bamlss, http://bamlss.org.
- Umlauf, Klein, Simon, Zeileis (2021). *bamlss: A Lego Toolbox for Flexible Bayesian Regression (and Beyond)* . Journal of Statistical Software, doi:10.18637/jss.v100.i04.