universität innsbruck

Flexible Distributional Regression Models: Methodology, Software, Applications

Nikolaus Umlauf

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

Overview

Joint work with Nadja Klein, Thomas Kneib, Stefan Lang, Thorsten Simon and Achim Zeileis.

Introduction

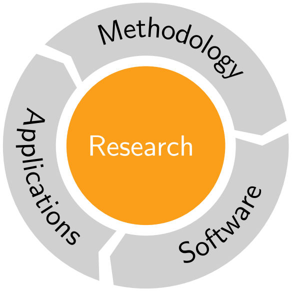
- **Computational power** has tremendously increased.
- **Complicated inferential problems**, e.g., with MCMC simulation, possible on virtually any modern computer.
- To embed many **different approaches** suggested in literature and software, a **unified modeling architecture** for flexible regression models is particularly helpful.
- With the *bamlss* framework, implementing (new) algorithms, integration of already existing software, is relatively straightforward.
- The original idea came from flexible **Bayesian distributional regression** models.

Introduction

Prerequisites:

- Very flexible regression framework,
- computational intensive,
- implementation is not straightforward.

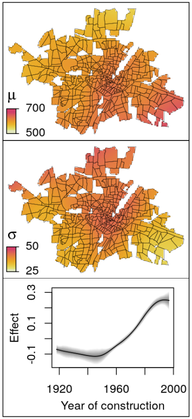
Extensions usually application based, on the edge of what is possible.



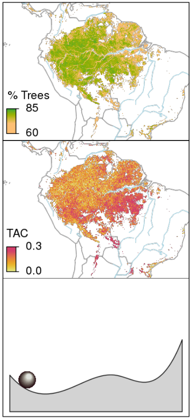
Introduction

Applications

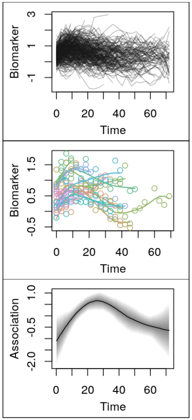
Real estate



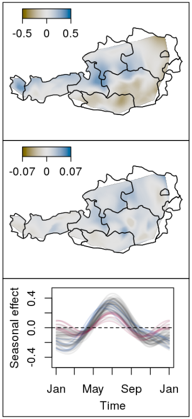
Remote sensing



Medicine



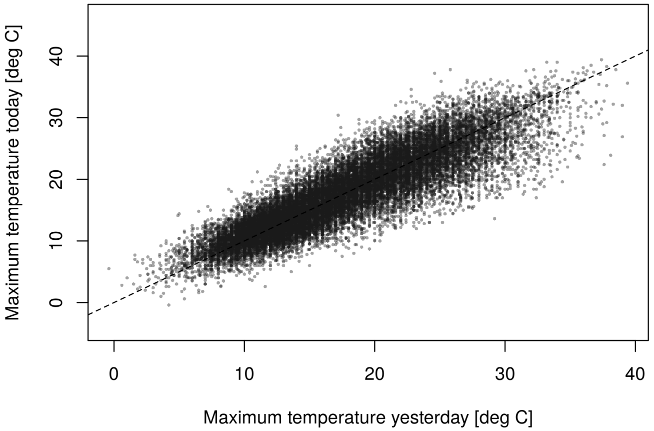
Meteorology



Introduction

Santiago de Compostela daily max. T (1944/11-2018/12).

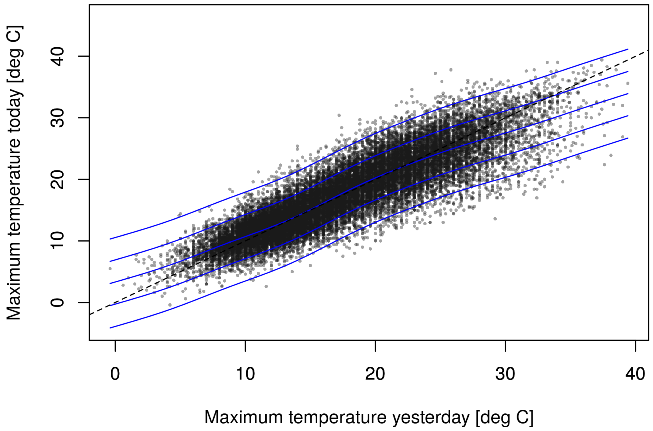
$$\mathtt{T} \sim N(\mu, \sigma^2).$$



Introduction

Santiago de Compostela daily max. T (1944/11-2018/12).

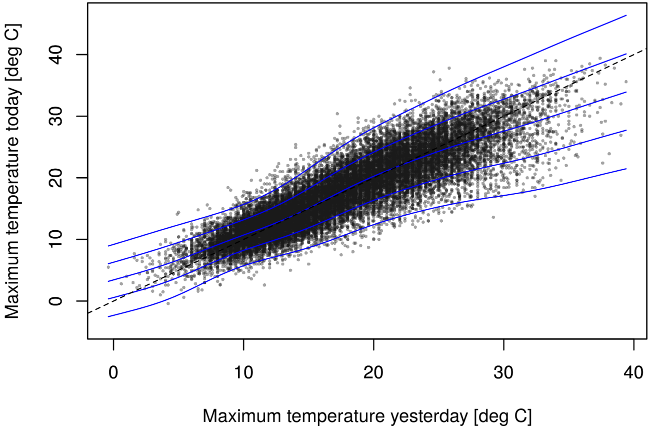
$$\mathtt{T} \sim N(\mu = f(\mathtt{T}_{t-1}), \, log(\sigma^2) = \beta_0).$$



Introduction

Santiago de Compostela daily max. T (1944/11-2018/12).

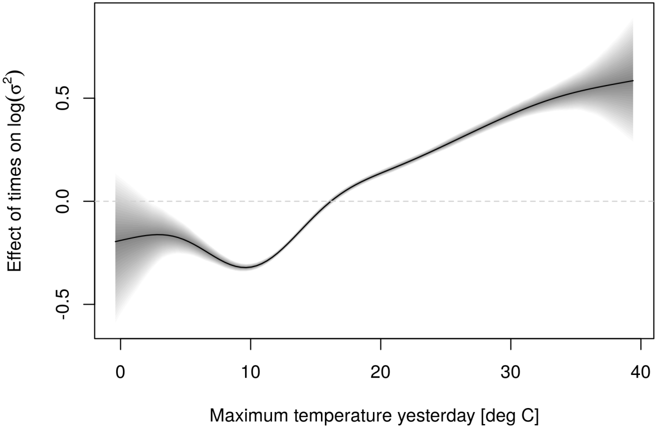
$$\mathtt{T} \sim N(\mu = f(\mathtt{T}_{t-1}),\ log(\sigma^2) = f(\mathtt{T}_{t-1})).$$



Introduction

Santiago de Compostela daily max. T (1944/11-2018/12).

$$\mathtt{T} \sim N(\mu = f(\mathtt{T}_{t-1}),\ log(\sigma^2) = f(\mathtt{T}_{t-1})).$$



Model specification

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

$$y \sim \mathcal{D}\left(h_1(\theta_1) = \eta_1,\ h_2(\theta_2) = \eta_2, \ldots,\ h_K(\theta_K) = \eta_K\right),$$

Each parameter is linked to a structured additive predictor

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

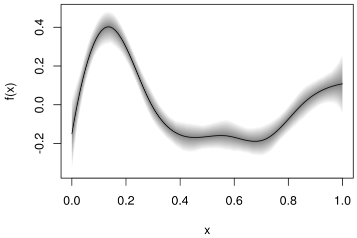
$j = 1, \ldots, J_k$ and $k = 1, \ldots, K$ and $h_k(\cdot)$ are link functions.

Vector of function evaluations $\mathbf{f}_{jk} = (f_{jk}(\mathbf{x}_1; \boldsymbol{\beta}_{jk}), \ldots, f_{jk}(\mathbf{x}_n; \boldsymbol{\beta}_{jk}))^\top$

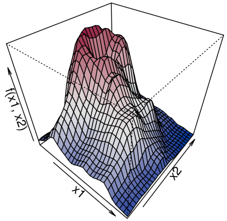
$$\mathbf{f}_{jk} = \begin{pmatrix} f_{jk}(\mathbf{x}_1; \boldsymbol{\beta}_{jk}) \\ \vdots \\ f_{jk}(\mathbf{x}_n; \boldsymbol{\beta}_{jk}) \end{pmatrix} = f_{jk}(\mathbf{X}_{jk}; \boldsymbol{\beta}_{jk}).$$

Model specification

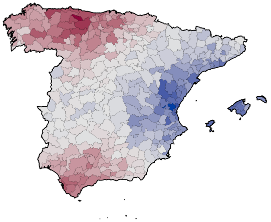
Nonlinear effects of continuous covariates



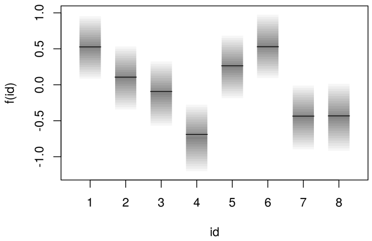
Two-dimensional surfaces



Spatially correlated effects f(x) = f(s)

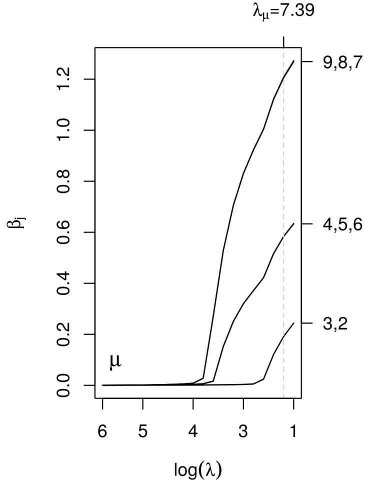


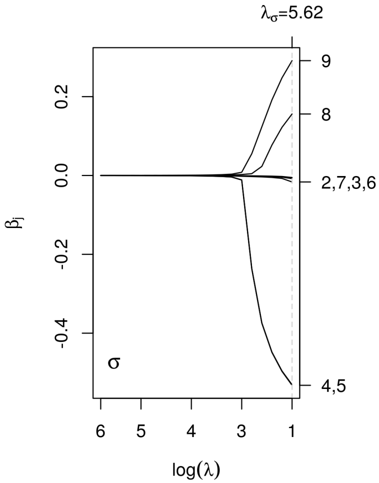
Random intercepts f(x) = f(id)



Model specification

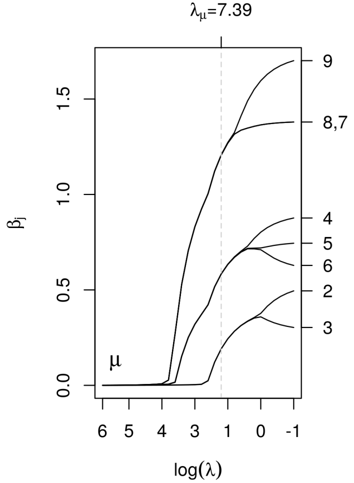
Model terms $f_{jk}(\mathbf{x}; \beta_{jk})$ with LASSO-type penalties $J_c(\beta_{jk})$.

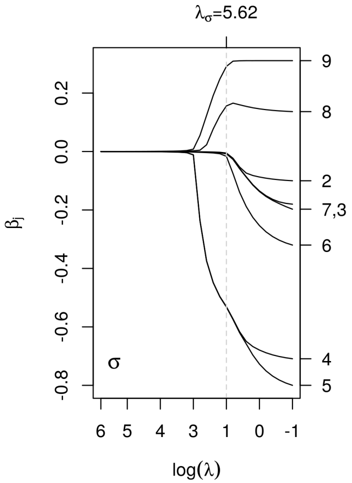




Model specification

Model terms $f_{jk}(\mathbf{x}; \beta_{jk})$ with LASSO-type penalties $J_f(\beta_{jk})$.





Model fitting

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

$$\begin{aligned}\ell(\boldsymbol{\beta}; \mathbf{y}, \mathbf{X}) &= \sum_{i=1}^{n} \log\ d_y(y_i; \theta_{i1} = h_1^{-1}(\eta_{i1}(\mathbf{x}_i, \boldsymbol{\beta}_1)), \ldots \\ &\qquad \ldots, \theta_{iK} = h_K^{-1}(\eta_{iK}(\mathbf{x}_i, \boldsymbol{\beta}_K))),\end{aligned}$$

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

The log-posterior

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

where $\boldsymbol{\tau} = (\boldsymbol{\tau}_1^\top, \ldots, \boldsymbol{\tau}_K^\top)^\top = (\boldsymbol{\tau}_{11}^\top, \ldots, \boldsymbol{\tau}_{J_1 1}^\top, \ldots, \boldsymbol{\tau}_{1K}^\top, \ldots, \boldsymbol{\tau}_{J_K K}^\top)^\top$ (frequentist, penalized log-likelihood).

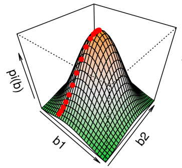
Model fitting

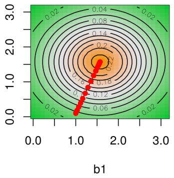
Bayesian point estimates of parameters are obtained by:

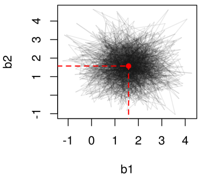
1. Maximization of the log-posterior for posterior mode estimation.
2. Solving high dimensional integrals, e.g., for posterior mean or median estimation.

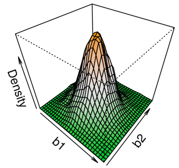
Problems 1 and 2 are commonly solved by computer intensive iterative algorithms of the following type:

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









Model fitting

Fortunately, partitioned updating is possible.

A simple generic algorithm for flexible regression models:

```
while(eps > $\varepsilon$ & t < maxit) {
    for(k in 1:K) {
        for(j in 1:J[k]) {
            Compute $\tilde{\boldsymbol{\eta}} = \boldsymbol{\eta}_{\mathtt{k}} - \mathbf{f}_{\mathtt{jk}}$.
            Obtain new $(\boldsymbol{\beta}^{\star}_{\mathtt{jk}}, \boldsymbol{\tau}^{\star}_{\mathtt{jk}})^{\top} = \mathtt{U}_{\mathtt{jk}}(\mathbf{X}_{\mathtt{jk}}, \mathbf{y}, \tilde{\boldsymbol{\eta}}, \boldsymbol{\beta}^{[\mathtt{t}]}_{\mathtt{jk}}, \boldsymbol{\tau}^{[\mathtt{t}]}_{\mathtt{jk}}, \boldsymbol{\alpha}_{jk})$.
            Update $\boldsymbol{\eta}_{\mathtt{k}} = \tilde{\boldsymbol{\eta}} + \mathbf{f}^{\star}_{\mathtt{jk}}$.
        }
    }
    t = t + 1
    Compute new eps.
}
```

Functions $\mathtt{U}_{\mathtt{jk}}(\cdot)$ could either return updates from an optimizing algorithm or proposals from a MCMC sampler.

Model fitting

MCMC simulation:

- Random walk Metropolis, symmetric $q(\boldsymbol{\beta}^{\star}_{jk}|\boldsymbol{\beta}^{(t)}_{jk})$.
- Derivative based MCMC, second order Taylor series expansion centered at the last state $\pi(\boldsymbol{\beta}^{\star}_{jk}|\cdot)$ yields $\mathcal{N}(\boldsymbol{\mu}^{(t)}_{jk}, \boldsymbol{\Sigma}^{(t)}_{jk})$ proposal with

$$\begin{aligned}\left(\boldsymbol{\Sigma}^{(t)}_{jk}\right)^{-1} &= -\mathbf{H}_{kk}\left(\boldsymbol{\beta}^{(t)}_{jk}\right)\\ \boldsymbol{\mu}^{(t)}_{jk} &= \boldsymbol{\beta}^{(t)}_{jk} - \mathbf{H}_{kk}\left(\boldsymbol{\beta}^{(t)}_{jk}\right)^{-1}\mathbf{s}\left(\boldsymbol{\beta}^{(t)}_{jk}\right).\end{aligned}$$

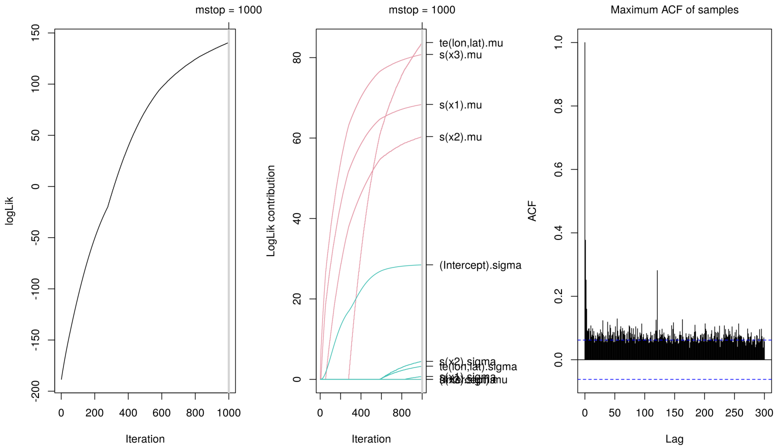
  Metropolis-Hastings acceptance probability

$$\alpha\left(\boldsymbol{\beta}^{\star}_{jk}|\boldsymbol{\beta}^{(t)}_{jk}\right) = \min\left\{\frac{p(\boldsymbol{\beta}^{\star}_{jk}|\cdot)q(\boldsymbol{\beta}^{(t)}_{jk}|\boldsymbol{\beta}^{\star}_{jk})}{p(\boldsymbol{\beta}^{(t)}_{jk}|\cdot)q(\boldsymbol{\beta}^{\star}_{jk}|\boldsymbol{\beta}^{(t)}_{jk})}, 1\right\}.$$

- Other sampling schemes, e.g., slice sampling, NUTS, t-walk, ...?!

Model fitting

For complicated models use combination of algorithms, e.g., gradient boosting for finding starting values for MCMC.



Updating

Consider IWLS updating

$$\beta_{jk}^{(t+1)} = U_{jk}(\beta_{jk}^{(t)}; \cdot) = (\mathbf{X}_{jk}^{\top}\mathbf{W}_{kk}\mathbf{X}_{jk} + \mathbf{G}_{jk}(\boldsymbol{\tau}_{jk}))^{-1}\mathbf{X}_{jk}^{\top}\mathbf{W}_{kk}(\mathbf{z}_k - \boldsymbol{\eta}_{k,-j}^{(t+1)}).$$

Computational characteristics:

- Naive updating functions $U_{jk}(\cdot)$ not feasible for large data sets.
- Oftentimes nested data structures, e.g., observations within counties, counties within states.
- Slow mixing of Markov chains.
- Number of different observations smaller than sample size.
- Design matrices $\mathbf{X}_{jk}$ and "penalty matrices" $\mathbf{G}_{jk}(\boldsymbol{\tau}_{jk})$ are typically sparse.

Efficient Updating I

Typically the number of different observations $x_{(1)} < x_{(2)} < \cdots < x_{(m)}$ in $\mathbf{X}$ is much smaller than the total number n of observations, i.e., $m \ll n$. For **sorted** observations x_i:

- Index vector **ind** with $\textbf{ind}[i] \in \{1, \ldots, m\}$, i.e., if $x_i = x_{(s)}$ then $\textbf{ind}[i] = s$.
- Decompose the design matrix in $\mathbf{X} = \mathbf{D}\mathbf{P}\tilde{\mathbf{X}}$ where
- $\tilde{\mathbf{X}}$ is the $m \times L$ reduced design matrix for the different and sorted observations $x_{(1)}, \ldots, x_{(m)}$, i.e., $\tilde{\mathbf{X}}[s, l] = X_l(x_s)$, $s = 1, \ldots, m$, $l = 1, \ldots, L$,
- $\mathbf{P}$ is a $n \times L$ permutation matrix, which reverts the sorting, i.e., $\mathbf{P}[i, s] = I(\textbf{ind}[i] = s)$.
- $\mathbf{D}$ is a diagonal matrix, e.g., for varying coefficient models or $\mathbf{D} = \mathbf{I}$ for simple additive terms.
- For the function evaluations we obtain $\mathbf{f} = \mathbf{X}\beta = \mathbf{D}\mathbf{P}\tilde{\mathbf{X}}\beta$.

Efficient Updating I

Using the permutation, we get

$$\mathbf{X}_{jk}^\top \mathbf{W}_{kk} \mathbf{X}_{jk} = \tilde{\mathbf{X}}_{jk}^\top \mathbf{P}_{jk}^\top \mathbf{D}_{jk}^\top \mathbf{W}_{kk} \mathbf{D}_{jk} \mathbf{P}_{jk} \tilde{\mathbf{X}}_{jk} = \tilde{\mathbf{X}}_{jk}^\top \tilde{\mathbf{W}} \tilde{\mathbf{X}}_{jk},$$

where

$$\tilde{\mathbf{W}} = \mathbf{P}_{jk}^\top \mathbf{D}_{jk}^\top \mathbf{W}_{kk} \mathbf{D}_{jk} \mathbf{P}_{jk} = diag(\tilde{w}_1, \ldots, \tilde{w}_{m_{jk}})$$

and the "reduced" weights $\tilde{w}_s$, are given by

$$\tilde{w}_s = \sum_{i\,:\,\mathbf{ind}[i]=s} z_i^2 \mathbf{W}_{kk}[i,i].$$

The weights $\tilde{w}_s$ can be computed by first initializing $\tilde{w}_s = 0$ followed by a simple loop:

For $i = 1, \ldots, n$ add $z_i^2 \mathbf{W}_{kk}[i,i]$ to $\tilde{w}_{\mathbf{ind}[i]}$.

Efficient Updating I

For $\mathbf{X}_{jk}^\top \mathbf{W}_{kk}(\mathbf{z}_k - \boldsymbol{\eta}_{k,-j}^{(t+1)})$ we obtain

$$\mathbf{X}_{jk}^\top \mathbf{W}_{kk}\mathbf{r} = \tilde{\mathbf{X}}_{jk}^\top \mathbf{P}_{jk}^\top \mathbf{D}_{jk}^\top \mathbf{W}_{kk}\mathbf{r} = \tilde{\mathbf{X}}_{jk}^\top \tilde{\mathbf{r}},$$

with partial residuals $\mathbf{r} = \mathbf{z}_k - \boldsymbol{\eta}_{k,-j}^{(t+1)}$.

The "reduced" partial residuals yield a $m_{jk} \times 1$ vector $\tilde{\mathbf{r}} = (\tilde{r}_1, \ldots, \tilde{r}_{m_{jk}})^\top$ given by

$$\tilde{r}_s = \sum_{i\,:\,\mathbf{ind}[i]=s} z_i\, \mathbf{W}_{kk}[i,i]\, r_i.$$

The $\tilde{r}_s$ are computed by first initializing $\tilde{r}_s = 0$ followed by the loop:

For $i = 1, \ldots, n$ add $z_i \mathbf{W}_{kk}[i,i]\, r_i$ to $\tilde{r}_{\mathbf{ind}[i]}$.

Efficient Updating I

Example using simulated data.

```
R> d <- GAMart(n = 10000)
R> d$x1 <- round(d$x1, 2)
R> X <- smooth.construct(s(x1, bs = "ps", k = 22), d, NULL)$X
R> dim(X)

[1] 10000    22

R> i <- match.index(X)
R> tX <- X[i$nodups, ]
R> dim(tX)

[1] 101  22

R> print(object.size(X), units = "Mb")

1.7 Mb

R> print(object.size(tX), units = "Kb")

17.6 Kb
```

Sparsity

B-spline penalty matrix:

$$\mathbf{K}_{jk} = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix}$$

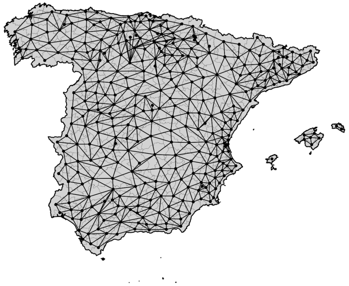
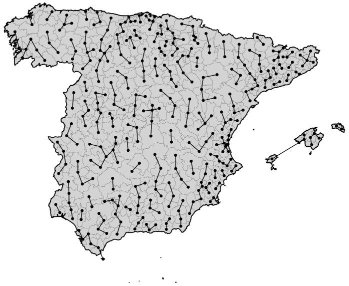
Sparsity

Markov random fields (MRF) design matrix:

$$\mathbf{X}_{jk} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

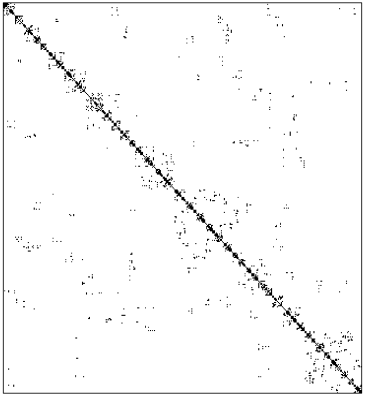
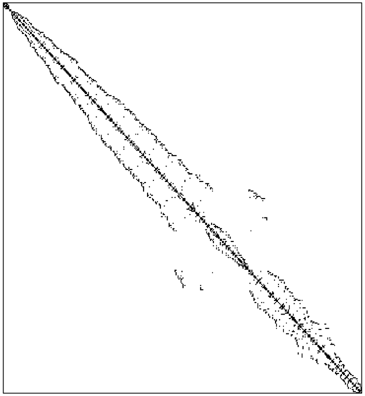
Sparsity

MRF penalty matrices are build using neighborhood structures.

Boundary neighbors	K-nearest neighbors
	

Sparsity

MRF penalty matrices are build using neighborhood structures.

Unsorted	Sorted, reverse Cuthill-McKee
	

Efficient Updating II

Products $\tilde{\mathbf{X}}_{jk}^{\top}\tilde{\mathbf{W}}\tilde{\mathbf{X}}_{jk}$ and $\tilde{\mathbf{X}}_{jk}^{\top}\tilde{\mathbf{r}}$ are stored in sparse matrix format.

Nonzero entries are stored in a vector $\mathbf{C}$ $(n_x \times 1)$. E.g., the l-th entry $\mathbf{C}[l]$ corresponds to

$$\mathbf{C}[l] = \sum_{s=1}^{m_{jk}} \tilde{w}_s \tilde{\mathbf{X}}_{jk}[s, r] \tilde{\mathbf{X}}_{jk}[s, l],$$

hence, most products are zero. Store the nonzero products in $\mathbf{h}_1$, the nonzero index s in $\mathbf{h}_2$ and the position of the first element in $\mathbf{h}_1$ in $\mathbf{h}_3$. Computation only requires

$$\mathbf{C}[l] = \sum_{s=\mathbf{h}_3[l]}^{\mathbf{h}_3[l+1]-1} \tilde{w}_{\mathbf{h}_2[s]} \mathbf{h}_1[s].$$

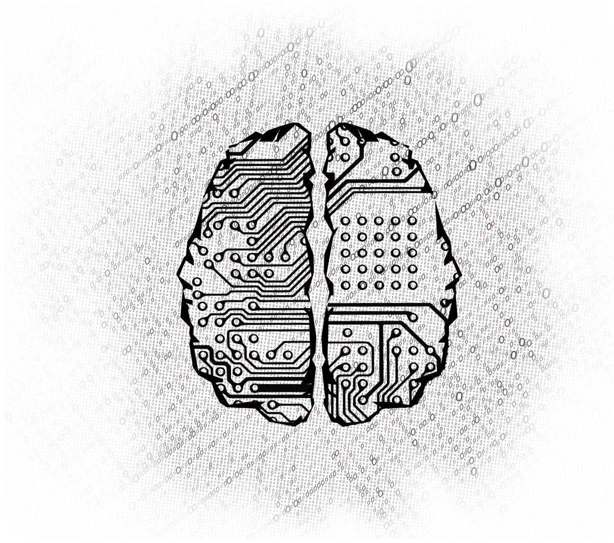
Similarly for $\tilde{\mathbf{X}}_{jk}^{\top}\tilde{\mathbf{r}}$, etc.

Efficient Updating II

Example using simulated data.

```
R> H <- sparse.matrix.index(tX)
R> print(head(H))
     [,1] [,2] [,3] [,4]
[1,]    6    7    8    9
[2,]   15   16   17   18
[3,]    8    9   10   11
[4,]   17   18   19   20
[5,]   18   19   20   21
[6,]    1    2    3    4
R> print(nrow(X) * ncol(X))
[1] 220000
R> print(nrow(tX) * ncol(tX))
[1] 2222
R> print(nrow(H) * ncol(H))
[1] 404
R> print(object.size(H), units = "Kb")
1.8 Kb
```

Neural Network Distributional Regression



Neural Network Distributional Regression

How to capture complex nonlinearities? Additive predictors $\eta_k(\mathbf{x}; \boldsymbol{\beta}_k)$ using regression splines have great performance, but can we do better?

- Feedforward neural networks (FNN) are extensively used in regression and classification applications.
- FNNs are universal function approximators (Hornik 1991).
- However, estimation is usually difficult and can involve thousands of parameters.
- Which makes the problem even harder in a full distributional regression setting (full Bayesian inference?).

$\Rightarrow$ Use FNN model term $f_{jk}(\mathbf{X}_{jk}; \boldsymbol{\beta}_{jk})$ additional to all other effects.

Neural Network Distributional Regression

Setup:

A FNN model term has a simple structure

$$f_{jk}(\mathbf{X}_{jk}; \boldsymbol{\beta}_{jk}) = \mathbf{X}_{jk}\boldsymbol{\beta}_{jk},$$

where the columns of $\mathbf{X}_{jk}$ are a decomposition of activation functions, e.g., using the sigmoid the l-th column (node) is

$$h_l(\mathbf{x}) = \frac{1}{1 + \exp(-(\mathbf{w}_l^\top \mathbf{x} + b_l))},$$

where $\mathbf{w}_l$ and b_l are inner weights and biases.

The activation function $h_l(\cdot)$ could also be Gauss (radial basis function network), sin, etc.

Neural Network Distributional Regression

Basic idea:

Reduce computational complexity, avoid non-convex optimization (time consuming, sensitive to initial values, local minima), by randomly selecting $\mathbf{w}_l$ and b_l, i.e., compute a random design matrix $\mathbf{X}_{jk}$.

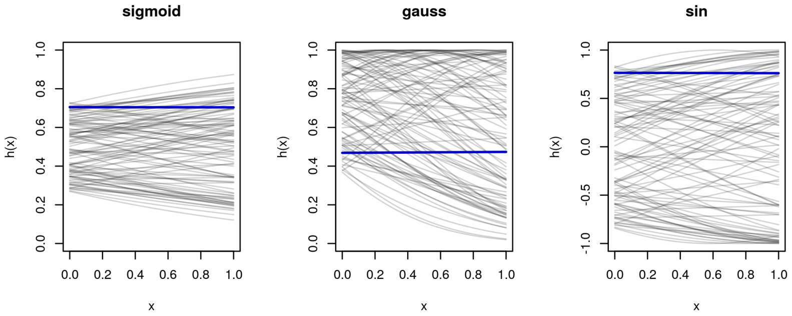
Although the idea is not new, this is now also known by the controversial name *extreme learning machine* (ELM, Huang 2006).

There are theoretical results that ELMs are also universal function approximators using symmetric intervals for the parameter scope (Husmeier 1999), a.o.

Neural Network Distributional Regression

Problems: How to randomly select $\mathbf{w}_l$ and b_l?

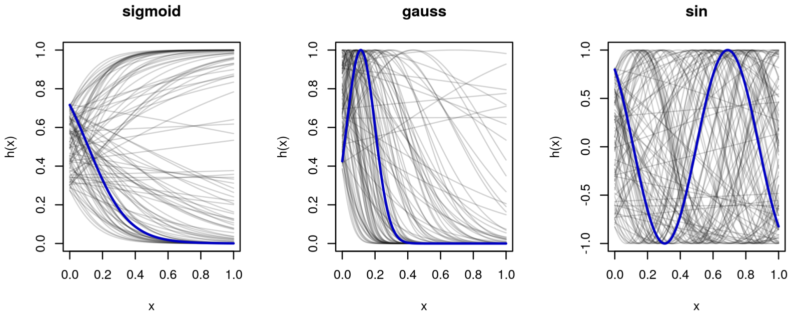
Sample $w_{ld}, b_l \sim \mathcal{U}(-1, 1)$. (Schmidt et al. 1992)



Neural Network Distributional Regression

Problems: How to randomly select $\mathbf{w}_l$ and b_l?

Sample $w_{ld} \sim \mathcal{U}(-10, 10)$ and $b_l \sim \mathcal{U}(-1, 1)$



Neural Network Distributional Regression

- Too small values for $\mathbf{w}_l$ and b_l lead to poor distribution of the basis functions (activation functions).
- Too large values will lead to saturated functions.
- Some literature about tuning the sampling range.
- Need a method that controls the flatness and steepness in the input hypercube.

$\Rightarrow$ Dudek (2017) gives a detailed description of how to select weights and biases for different activation functions.

Neural Network Distributional Regression

Sampling weights: Dudek (2017)

For $[0,1]$ scaled inputs, weights are sampled such that the most nonlinear and steepest parts are inside the data region.

1. Given r and s, sample sum of input weights

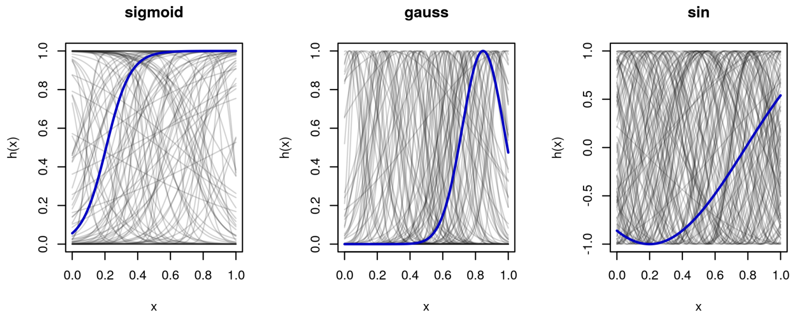
$$\sum\nolimits_{[l]} \sim \mathcal{U}\left(\log\left[\frac{1-r}{r}\right], s \cdot \log\left[\frac{1-r}{r}\right]\right).$$

2. For $\mathbf{w}_l$ sample $\zeta_d \sim \mathcal{U}(-1,1)$.
3. Set $w_{ld} = \zeta_d \frac{\sum_{[l]}}{\sum_d \zeta_d}$.
4. Set $b_l = -\sum_d w_{ld} z_l$, where $z_l \sim \mathcal{U}(0,1)$.

Depending on the activation functions, r and s can have different ranges.

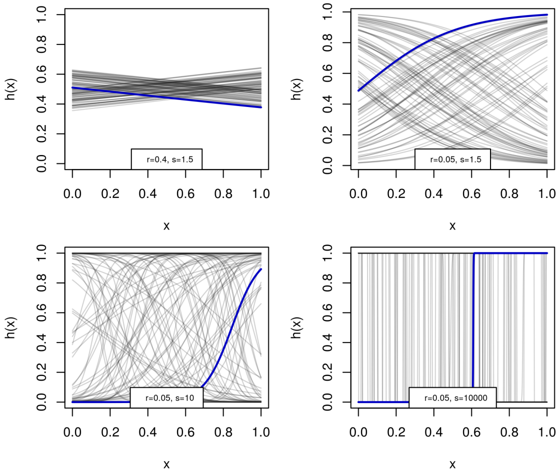
Neural Network Distributional Regression

Sampling weights: Dudek (2017)



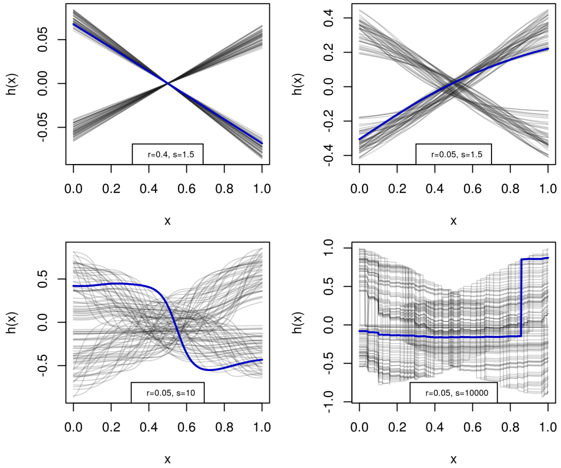
Neural Network Distributional Regression

Sampling weights: Scaling with r and s.



Neural Network Distributional Regression

Sampling weights: Centering.



Elastic net regularization

Overfitting:

We use elastic net regularization

$$\lambda_{jk1} \cdot J_{\mathsf{L}}(\boldsymbol{\beta}_{jk}) + \lambda_{jk2} \cdot J_{\mathsf{R}}(\boldsymbol{\beta}_{jk}),$$

with quadratic approximations of the LASSO penalties (compare Oelker & Tutz, 2017)

$$J_{\mathsf{L}}(\boldsymbol{\beta}_{jk}) \approx J_{\mathsf{L}}(\boldsymbol{\beta}_{jk}^{(t)}) + \frac{1}{2}\left(\boldsymbol{\beta}_{jk}^{\top}\mathbf{P}_{jk}(\boldsymbol{\beta}_{jk})\boldsymbol{\beta}_{jk} + (\boldsymbol{\beta}_{jk}^{(t)})^{\top}\mathbf{P}_{jk}(\boldsymbol{\beta}_{jk}^{(t)})\boldsymbol{\beta}_{jk}^{(t)}\right),$$

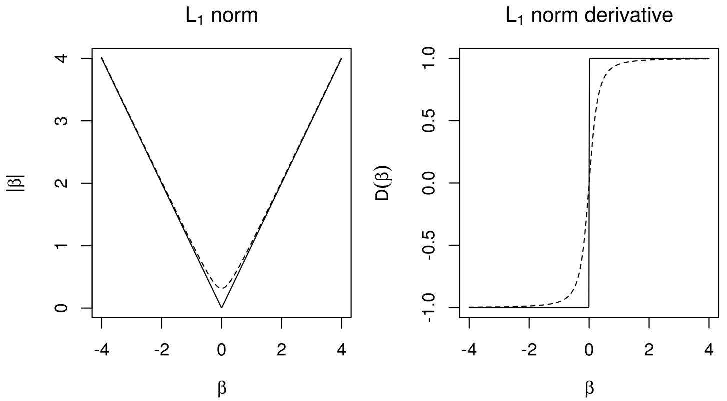
with

$$\mathbf{P}_{jk}(\boldsymbol{\beta}_{jk}^{(t)}) = q'_{jk}\left(\left\|\mathbf{a}_{jk}^{\top}\boldsymbol{\beta}_{jk}^{(t)}\right\|_{N_{jk}}\right) \cdot \frac{D_{jk}(\mathbf{a}_{jk}^{\top}\boldsymbol{\beta}_{jk}^{(t)})}{\mathbf{a}_{jk}^{\top}\boldsymbol{\beta}_{jk}^{(t)}} \cdot \mathbf{a}_{jk}\mathbf{a}_{jk}^{\top}.$$

E.g., $\|\beta\|_1 = |\beta|$ is approximated by $\sqrt{\beta^2 + c}$, hence, IWLS based updating functions $\mathtt{U_{jk}}(\cdot)$ are relatively easy to implement.

Elastic net regularization

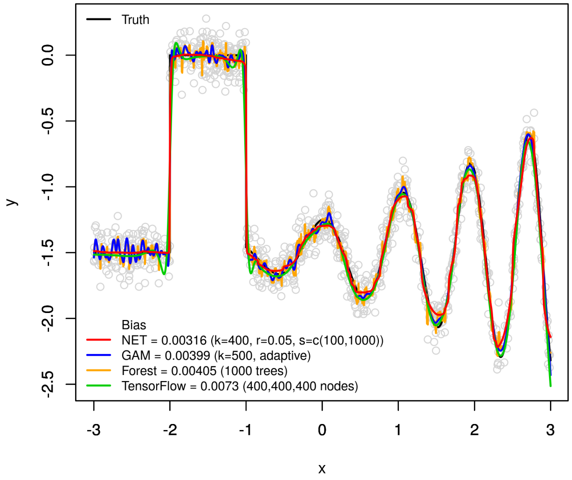
Example of the approximation of the L_1 norm.



Usually setting the constant to $c \approx 10^{-5}$ works well.

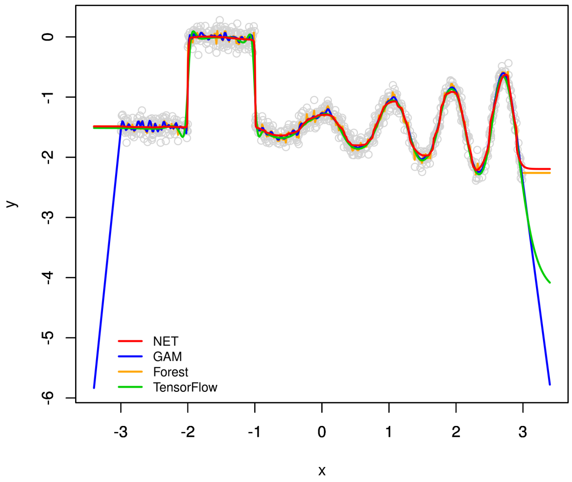
Neural Network Distributional Regression

Simulated example: Sigmoid activation.



Neural Network Distributional Regression

Simulated example: Out of range predictions.



R package *bamlss*

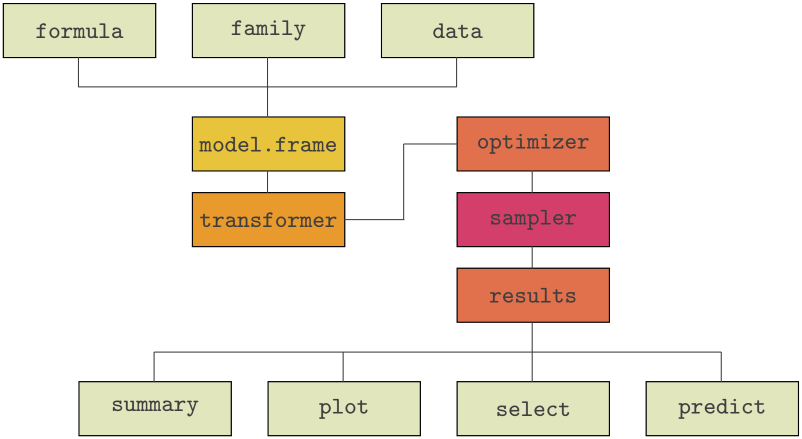
The package is available at

https://CRAN.R-project.org/package=bamlss

Development version, in R simply type

```
R> install.packages("bamlss",
+    repos = "http://R-Forge.R-project.org")
```

R package *bamlss*



In principle, the setup does not restrict to any specific type of engine (Bayesian or frequentist).

R package *bamlss*

Type	Function
Parser	`bamlss.frame()`
Transformer	`bamlss.engine.setup()`, `randomize()`
Optimizer	`bfit()`, `opt()`, `cox.mode()`, `jm.mode()`
	`boost()`, `stabsel()`, `bboost()`, `lasso()`
Sampler	`GMCMC()`, `JAGS()`, `STAN()`, `BayesX()`,
	`cox.mcmc()`, `jm.mcmc()`
Results	`results.bamlss.default()`

To implement new engines, only the building block functions have to be exchanged.

R package *bamlss*

The package makes heavy uses of **mgcv infrastructures** using `smooth.construct()`, however, optimizers and samplers may use special model terms, e.g., the LASSO constructor `la()`.

```
R> f <- list(
+   num ~ s(x1) + s(x2) + la(id),
+   sigma ~ s(x1) + s(x2) + la(id)
+ )
R> bf <- bamlss.frame(f, data = d, family = "gaussian")
R> names(bf)

[1] "call"         "model.frame" "y"            "formula"
[5] "terms"        "family"      "x"

R> names(bf$x$mu)

[1] "formula"          "fake.formula"     "terms"
[4] "model.matrix"     "smooth.construct"

R> names(bf$x$mu$smooth.construct)

[1] "s(x1)"  "s(x2)"  "la(id)"
```

R package *bamlss*

Work in progress . . .

Function	Distribution
`beta_bamlss()`	Beta distribution
`binomial_bamlss()`	Binomial distribution
`cnorm_bamlss()`	Censored normal distribution
`cox_bamlss()`	Continuous time Cox-model
`gaussian_bamlss()`	Gaussian distribution
`gamma_bamlss()`	Gamma distribution
`gpareto_bamlss()`	Generalized Pareto distribution
`jm_bamlss()`	Continuous time joint-model
`multinomial_bamlss()`	Multinomial distribution
`mvn_bamlss()`	Multivariate normal distribution
`poisson_bamlss()`	Poisson distribution
. . .	

New families only require density, distribution, random number generator, quantile, score and hess functions. Wrapper for R package *gamlss* families.

R package *bamlss*

Wrapper function:

```
R> f <- list(y ~ la(id,fuse=2), sigma ~ la(id,fuse=1))
R> b <- bamlss(f, family = "gaussian", sampler = FALSE,
+    optimizer = lasso, criterion = "BIC", multiple = TRUE)
```

Standard extractor and plotting functions:

`summary()`, `plot()`, `fitted()`, `residuals()`, `predict()`, `coef()`, `logLik()`, `DIC()`, `samples()`, ...

R package *bamlss*

Example: model fitting functions.

```
bfit(x, y, family, start = NULL, weights = NULL, offset = NULL,
  update = "iwls", criterion = c("AICc", "BIC", "AIC"), ...)
```

```
boost(x, y, family, weights = NULL, offset = NULL,
  nu = 0.1, df = 4, maxit = 400, ...)
```

```
GMCMC(x, y, family, start = NULL, weights = NULL, offset = NULL,
  n.iter = 1200, burnin = 200, thin = 1, ...)
```

R package *bamlss*

Example: updating functions.

```
bfit_iwls(x, family, y, eta, id, weights, criterion, ...)


boost_fit(x, y, nu, hatmatrix = TRUE, weights = NULL, ...)


GMCMC_iwls(family, theta, id, eta, y, data,
  weights = NULL, offset = NULL, ...)


GMCMC_slice(family, theta, id, eta, y, data, ...)
```

Leukemia Survival Example

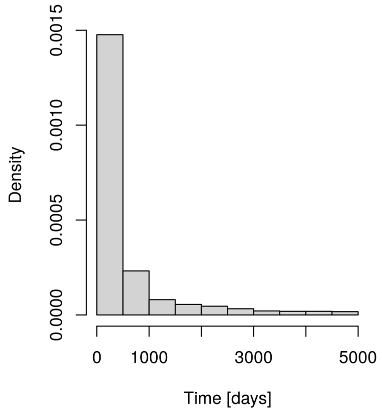
Data structure:

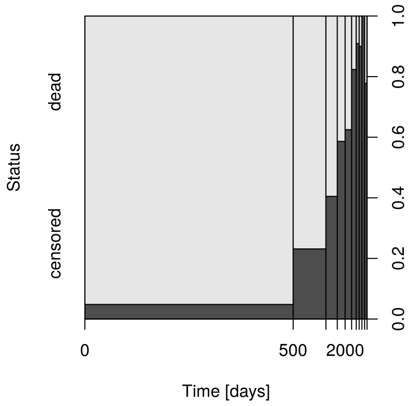
First analyzed by Henderson et al. (2002), investigate spatial variation in survival after accounting for subject-specific factors in northwest England. ($n = 1043$ patients)

Variable	Description.
`time`	Survival time in days.
`cens`	Right censoring status 0=censored, 1=dead.
`xcoord`	Coordinates in x-axis of residence.
`ycoord`	Coordinates in y-axis of residence.
`age`	Age in years.
`sex`	male=1 female=0.
`wbc`	White blood cell count at diagnosis, truncated at 500.
`tpi`	The Townsend score for which higher values indicates less affluent areas.
`district`	Administrative district of residence.

Leukemia Survival Example

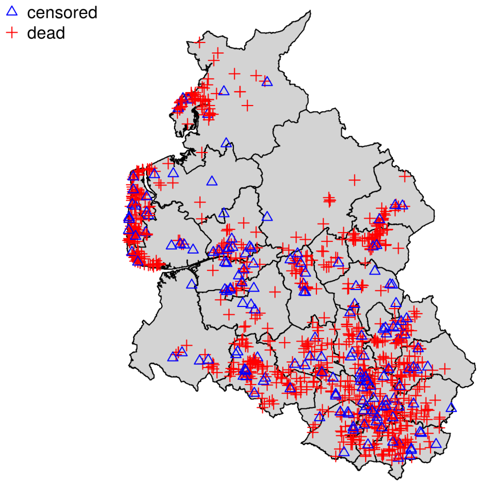
Survival times:





Leukemia Survival Example

Spatial distribution:



Leukemia Survival Example

Cox model:

The hazard of an event (status `dead`) at time t can be described with a relative additive risk model of the form:

$$\lambda(t) = \exp\left(\eta(t)\right) = \exp\left(\eta_\lambda(t) + \eta_\gamma\right),$$

i.e., a model for the instantaneous risk conditional on being alive before time t.

The probability to not survive after time t is

$$S(t) = Prob(T > t) = \exp\left(-\int_0^t \lambda(u)du\right).$$

Leukemia Survival Example

For NR and MCMC we need the log-likelihood of the continuous time Cox-model

$$\ell(\boldsymbol{\beta}; \mathbf{y}, \mathbf{X}) = \sum_{i=1}^{n} \left(\delta_i \eta_{i,\gamma} - \int_0^{t_i} \exp(\eta_{i,\lambda}(u) du) \right)$$

Assuming a basis function approach, the score vector for the time-dependent part is

$$\mathbf{s}(\boldsymbol{\beta}_\lambda) = \boldsymbol{\delta}^\top \mathbf{X}_\lambda(\mathbf{t}) - \sum_{i=1}^{n} \exp(\eta_{i,\gamma}) \left(\int_0^{t_i} \exp(\eta_{i,\lambda}(u)) \mathbf{x}_i(u) du \right).$$

The elements of the Hessian w.r.t. $\boldsymbol{\beta}_\lambda$ are

$$\mathbf{H}(\boldsymbol{\beta}_\lambda) = -\sum_{i=1}^{n} \exp(\eta_{i,\gamma}) \int_0^{t_i} \exp(\eta_{i,\lambda}(u)) \mathbf{x}_{i,\lambda}(u) \mathbf{x}_{i,\lambda}^\top(u) du.$$

Leukemia Survival Example

The integrals need to be computed numerically, e.g., using the trapezoidal rule we “only” need to set up a time grid, lets say with 100 equidistant points within $[0, t_i]$

$$\mathbf{G} = \begin{pmatrix} \mathbf{g}_1^\top \\ \vdots \\ \mathbf{g}_n^\top \end{pmatrix}, \quad \text{with} \quad \mathbf{g}_i = (0, \dots, t_i)^\top,$$

to construct the evaluated $\lambda(t)$ matrix with

$$\hat{\eta}_\lambda(\mathbf{G}) = \begin{pmatrix} \sum_{j=1}^{J_\lambda} f_j(x_{1j}(g_{10})) & \dots & \sum_{j=1}^{J_\lambda} f_j(x_{1j}(g_{1t_i})) \\ \vdots & \ddots & \vdots \\ \sum_{j=1}^{J_\lambda} f_j(x_{nj}(g_{n0})) & \dots & \sum_{j=1}^{J_\lambda} f_j(x_{nj}(g_{nt_i})) \end{pmatrix}.$$

Leukemia Survival Example

Fortunately, the time-constant part is a bit easier. Results in IWLS backfitting/proposal scheme with

$$\mathbf{z} = \boldsymbol{\eta}_\gamma + \mathbf{W}^{-1}\mathbf{u}$$

with diagonal matrix

$$\mathbf{W} = \mathit{diag}(\exp(\boldsymbol{\eta}_\gamma) \cdot \mathbf{I})$$

and

$$\mathbf{u} = \boldsymbol{\delta} - \exp(\boldsymbol{\eta}_\gamma) \cdot \mathbf{I}.$$

Here, diagonal matrix $\mathbf{I}$ represents the integrals for all individuals.

Optimizer and sampler implemented in function `cox.mode()` and `cox.mcmc()`.

Leukemia Survival Example

For the leukemia survival example, we use the following additive predictors

$$\eta_\lambda = f_1(\texttt{time}) + f_2(\texttt{time}, \texttt{sex}, \texttt{age}, \texttt{wbc}, \texttt{tpi}, \texttt{xcoord}, \texttt{ycoord})$$

and

$$\begin{aligned}\eta_\gamma &= \beta_0 + \texttt{sex} + f_3(\texttt{age}) + f_4(\texttt{wbc}) + f_5(\texttt{tpi}) + \\ &\quad f_6(\texttt{xcoord}, \texttt{ycoord}) + \\ &\quad f_7(\texttt{sex}, \texttt{age}, \texttt{wbc}, \texttt{tpi}, \texttt{xcoord}, \texttt{ycoord}).\end{aligned}$$

Here, functions $f_2(\cdot)$ and $f_7(\cdot)$ represent a time dependent and a time constant neural network model term.

For the other functions we use regression splines.

Leukemia Survival Example

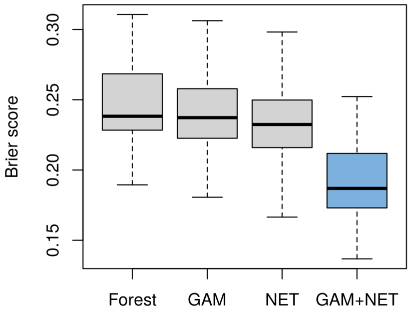
In R we set up the model by

```
R> library("bamlss")
R> library("survival")
R> data("LeukSurv", package = "spBayesSurv")
R> ftd <- ~ time + sex + age + wbc + tpi + xcoord + ycoord
R> ftc <- ~ sex + age + wbc + tpi + xcoord + ycoord
R> f <- list(
+   Surv(time, cens) ~ s(time) +
+     n(ftd,k=300,pt="lasso",
+       rint=list("sigmoid"=0.1,"gauss"=0.1),
+       sint=list("sigmoid"=c(5,10),"gauss"=5),
+       afun=c("sigmoid","gauss"),ndf=50),
+   gamma ~ sex + s(age) + s(wbc) + s(tpi) + s(xcoord,ycoord,k=100) +
+     n(ftc,k=300,pt="lasso",
+       rint=list("sigmoid"=0.1,"gauss"=0.1),
+       sint=list("sigmoid"=c(5,10),"gauss"=5),
+       afun=c("sigmoid","sin","gauss"),ndf=50)
+ )
R> b <- bamlss(f, data = LeukSurv, family = "cox")
```

Leukemia Survival Example

Performance:

We evaluate the performance of the neural network Cox model by randomly sampling 100 individuals that serve as a hold out sample and compare using the Brier score. This is done 50 times.



In sample Brier score: GAM=0.24, GAM+NET=0.18.

Leukemia Survival Example

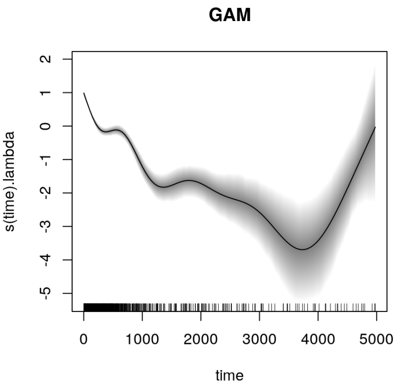
```
R> summary(b)

## Subset of full model summary.

Formula lambda:
---
Surv(time, cens) ~ s(time) + n(ftd, k = 300, pt = "lasso",
    rint = list(sigmoid = 0.1, gauss = 0.1),
    sint = list(sigmoid = c(5, 10), gauss = 5),
    afun = c("sigmoid", "gauss"), ndf = 50)

-
Smooth terms:
              parameters
s(time).tau21      0.000
s(time).edf        0.984
n(ftd).tau21      76.543
n(ftd).edf        34.061
---
```

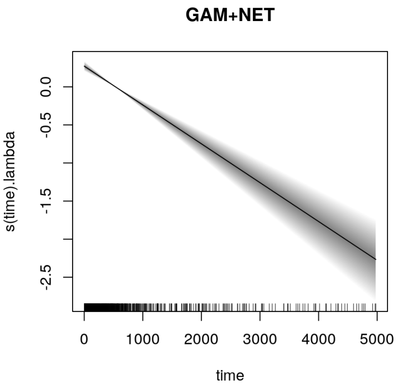
Leukemia Survival Example

```
Formula gamma:
---
gamma ~ sex + s(age) + s(wbc) + s(tpi) + s(xcoord, ycoord, k = 100) +
    n(ftc, k = 300, pt = "lasso", rint = list(sigmoid = 0.1,
        gauss = 0.1), sint = list(sigmoid = c(5, 10), gauss = 5),
        afun = c("sigmoid", "sin", "gauss"), ndf = 50)
-
Smooth terms:
                       parameters
s(age).tau21                0.000
s(age).edf                  0.997
s(wbc).tau21                0.000
s(wbc).edf                  0.977
s(tpi).tau21               86.135
s(tpi).edf                  7.954
s(xcoord,ycoord).tau21      0.147
s(xcoord,ycoord).edf        7.935
n(ftc).tau21                0.000
n(ftc).edf                  0.000
---
```

Leukemia Survival Example

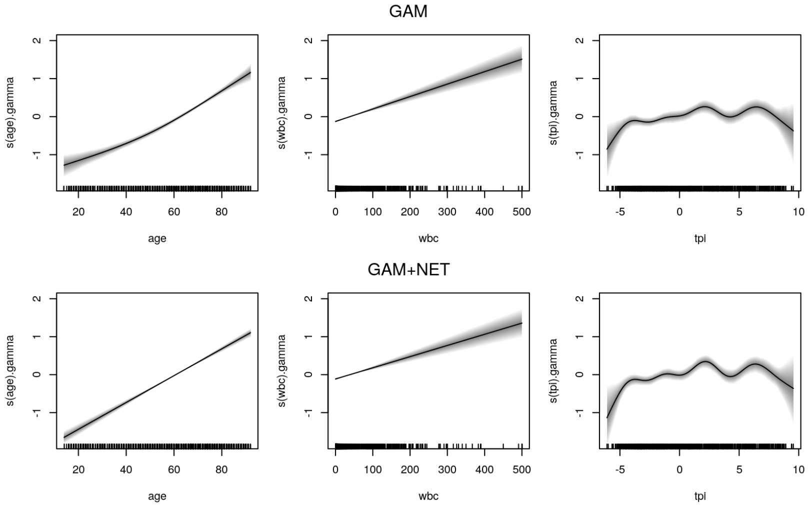
```
R> plot(b, model = "lambda", term = "s(time)")
```





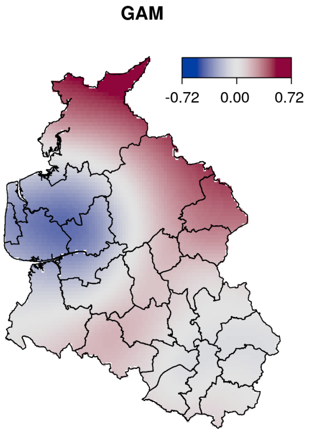
Leukemia Survival Example

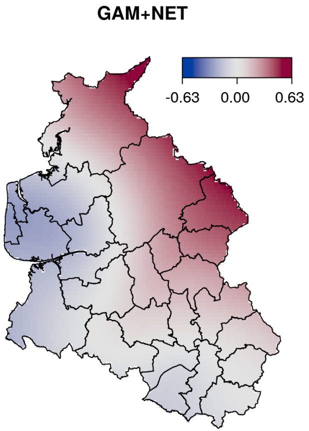
```
R> plot(b, model = "gamma", term = c("s(age)", "s(wbc)", "s(tpi)"))
```



Leukemia Survival Example

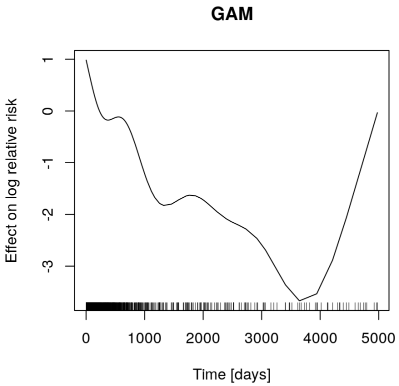
```
R> predict(b, newdata = nd,
+    model = "gamma", term = "s(xcoord,ycoord)")
```

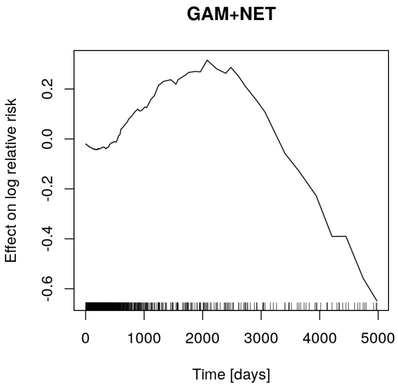




Leukemia Survival Example

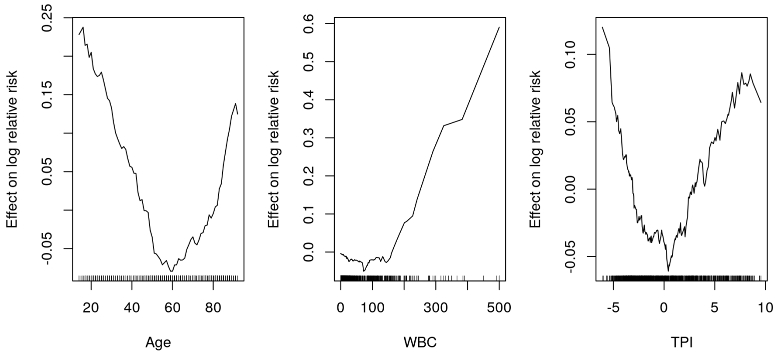
Accumulated local effects (ALE) plots: (Apley D.W., 2016)





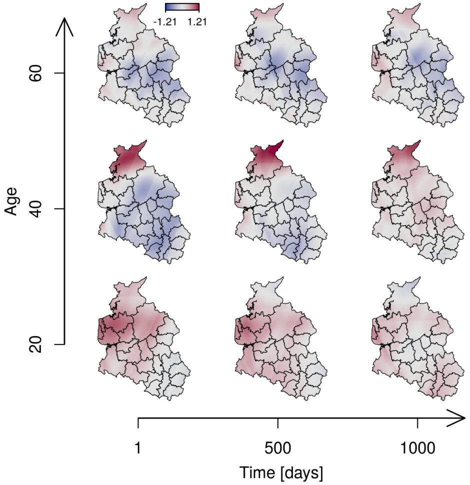
Leukemia Survival Example

Accumulated local effects (ALE) plots: (Apley D.W., 2016)



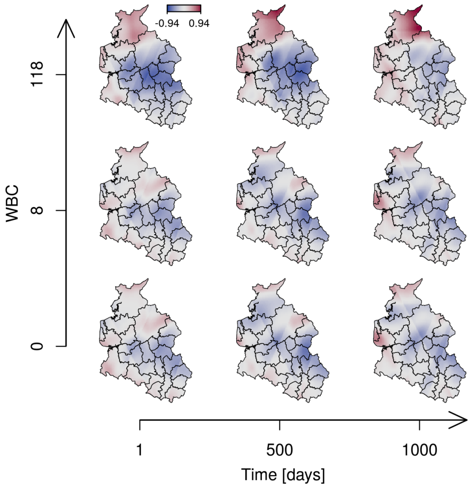
Leukemia Survival Example

Interaction plots: (females, remaining variables fixed at means)



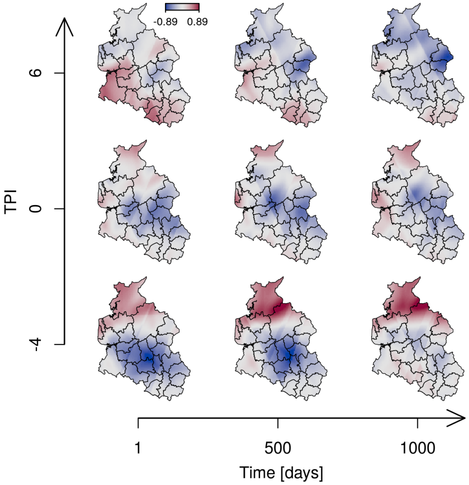
Leukemia Survival Example

Interaction plots: (females, remaining variables fixed at means)



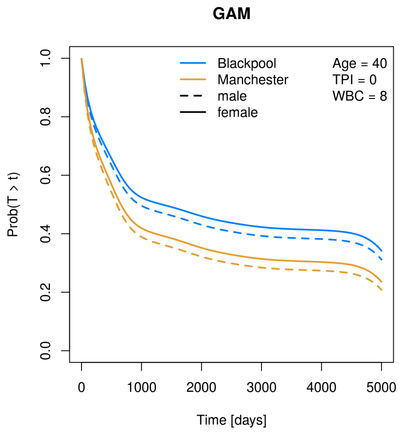
Leukemia Survival Example

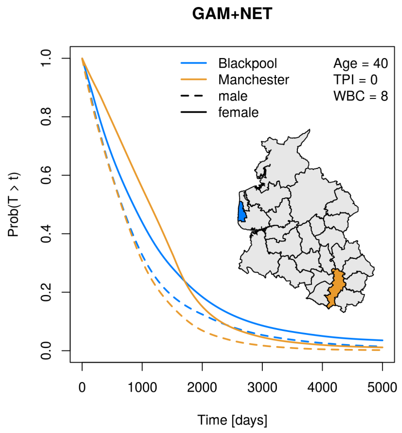
Interaction plots: (females, remaining variables fixed at means)



Leukemia Survival Example

Probabilities: Blackpool vs. Manchester.

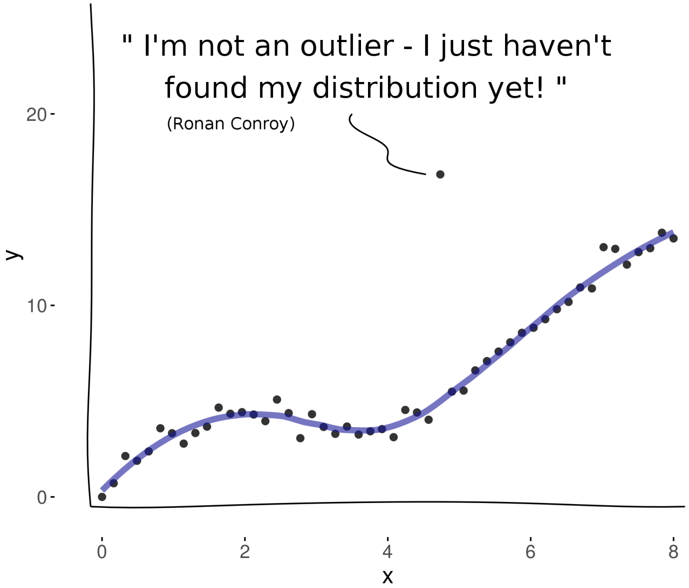




Summary & Outlook

- Neural networks really seem to have good approximation skills.
- Capable to find high-order interactions.
- However, this needs to be further investigated.
- Good predictive performance, but interpretation is still difficult.

- Linears vs. nonlinear direct connectors?
- Tune weights instead of random sampling?
- Full Bayesian inference for weights?
- Deep networks?


" I'm not an outlier - I just haven't found my distribution yet! "
(Ronan Conroy)
y
20
10
0
0
2
4
6
8
x

References & Software

Apley D.W. (2016). Visualizing the effects of predictor variables in black box supervised learning models. arXiv:1612.08468.

Dudek G. (2017). A method of generating random weights and biases in feedforward neural networks with random hidden nodes. arXiv:1710.04874.

Groll A., Hambuckers J., Kneib T. & and Umlauf N. (2018). Lasso-type penalization in the framework of generalized additive models for location, scale and shape. Working papers, Faculty of Economics and Statistics, University of Innsbruck. `https://econpapers.repec.org/paper/innwpaper/2018-16.htm`.

Henderson R., Shimakura S. & and Gorst D. (2002). Modeling spatial variation in leukemia survival data. *Journal of the American Statistical Association* **70**(460).

Hornik K. (1991). Approximation capabilities of multilayer feedforward networks *Neural Networks* **4(2)**, 251 – 257.

Huang G.B., Zhu Q.Y. & Siew, C.K. (2006). Extreme Learning Machine: Theory and Applications. *Neurocomputing* **70**, 489 – 501.

Husmeier D. (1999). Random vector functional link (RVFL) networks. *Neural Networks for Conditional Probability Estimation: Forecasting Beyond Point Predictions*, Chapter 6, Springer.

Lang S., Umlauf N., Wechselberger P., Harttgen K. & and Kneib T. (2012). Multilevel structured additive regression. *Statistics and Computing*.
URL: `http://link.springer.com/article/10.1007/s11222-012-9366-0`

References & Software

Oelker M.R. & Tutz G. (2017). A uniform framework for the combination of penalties in generalized structured models. *Advances in Data Analysis and Classification* **11**(1), 97 – 120.

Rigby R.A. & Stasinopoulos D.M. (2005). Generalized additive models for location, scale and shape. *Journal of the Royal Statistical Society: Series C (Applied Statistics)* **54(3)**, 507 – 554.

Schmidt W.F., Kraaijveld M. & Duin R.P. (1992). Feedforward neural networks with random weights. *Proc. 11th IAPR IEEE Inter. Conf. on Pattern Recognition*, 1 – 4.

Stasinopoulos D.M., & Rigby R.A. (2007) Generalized additive models for location scale and shape (GAMLSS) in R. *Journal of Statistical Software* **23**(7).

Tibshirani R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society B 58* 267 – 288.

Umlauf N., Klein N. & Zeileis A. (2018). BAMLSS: Bayesian additive models for location, scale and shape (and beyond). Journal of Computational and Graphical Statistics, 2018. to appear. `doi:10.1080/10618600.2017.1407325`.

Umlauf N., Klein N., Zeileis A., Köhler M. & Thorsten S. (2019). **bamlss**: Bayesian additive models for location, scale and shape (and beyond). R package version 1.0-1, URL: `http://cran.r-project.org/package=bamlss`.