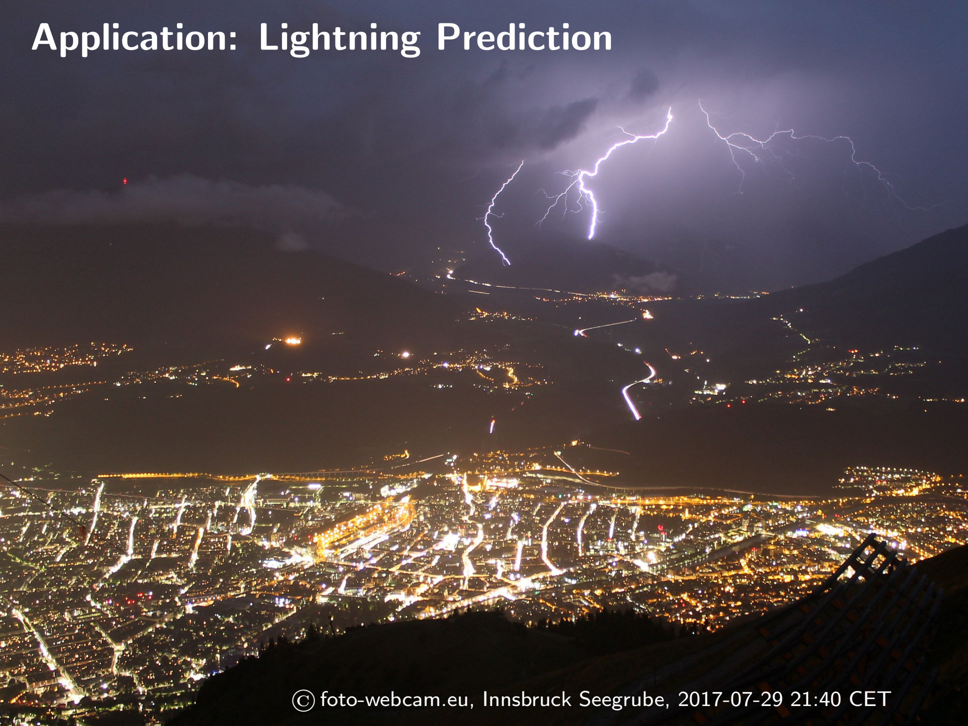
Flexible Distributional Regression Models for Very Large Datasets

Nikolaus Umlauf

http://nikum.org/

Application: Lightning Prediction



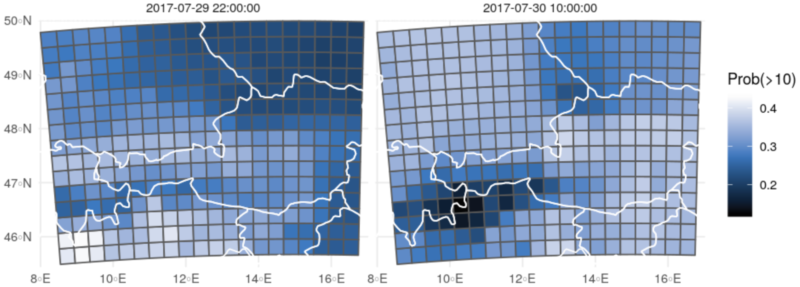
© foto-webcam.eu, Innsbruck Seegrube, 2017-07-29 21:40 CET

Forecasting Lightning

Goal: Forecast lightning by statistical post-processing of numerical weather prediction (NWP) output.

Step 1: Occurence. Is there any lightning? (Binary)

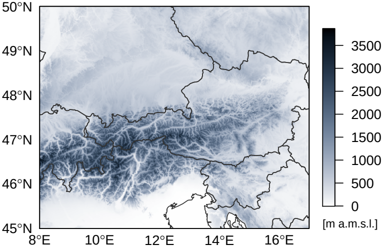
Step 2: Intensity. If there is any lightning, how many? (Counts > 0)



Data

ALDIS lightning counts:

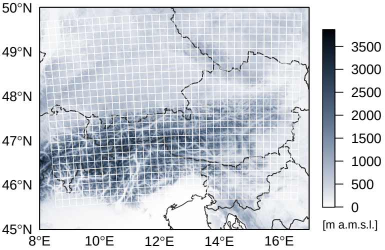
- Summer: May–August.
- Afternoons: 12–18 UTC.
- Gridded on 18×18 km^2.
- 2010–2017.
- #Obs. $\sim$ 8M.



Data

ALDIS lightning counts:

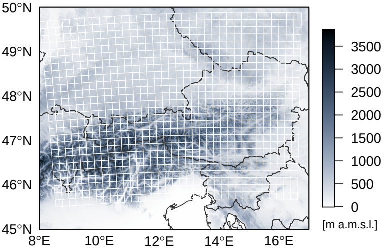
- Summer: May–August.
- Afternoons: 12–18 UTC.
- Gridded on 18×18 km^2.
- 2010–2017.
- #Obs. $\sim$ 8M.



Data

ALDIS lightning counts:

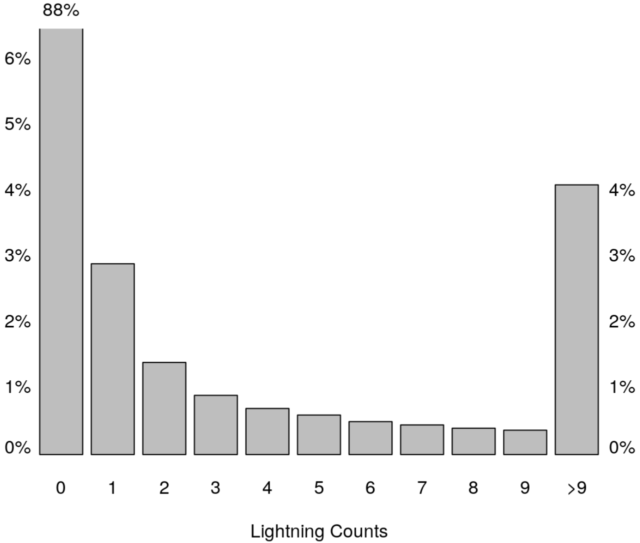
- Summer: May–August.
- Afternoons: 12–18 UTC.
- Gridded on 18×18 km^2.
- 2010–2017.
- #Obs. $\sim$ 8M.



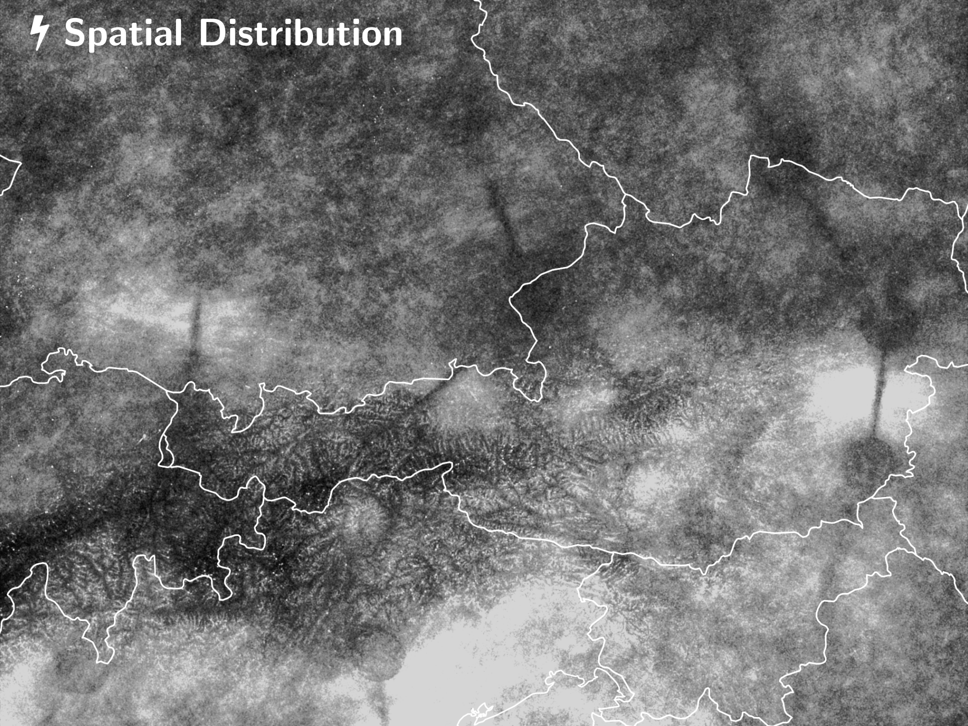
ECMWF ensemble forecasts:

- Forecast horizons: 1–5 days.
- 2010–2017.
- NWP outputs: Convective precipitation, CAPE, temperature, relative humidity, vertical velocity, radiation, heat fluxes, . . .
- Median and interquartile range.

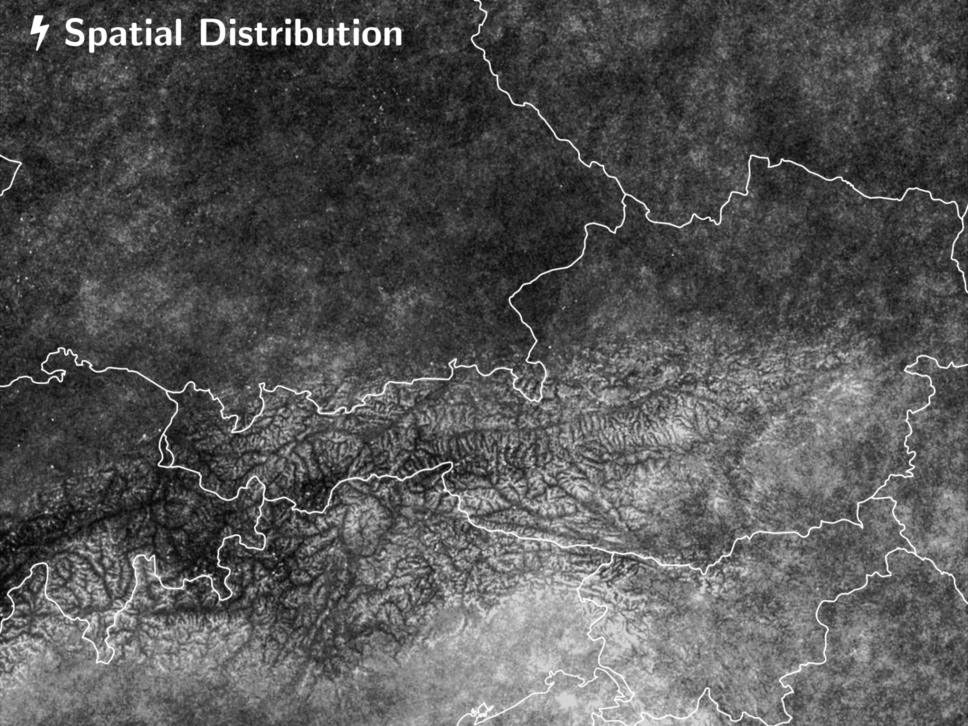
Lightning Counts



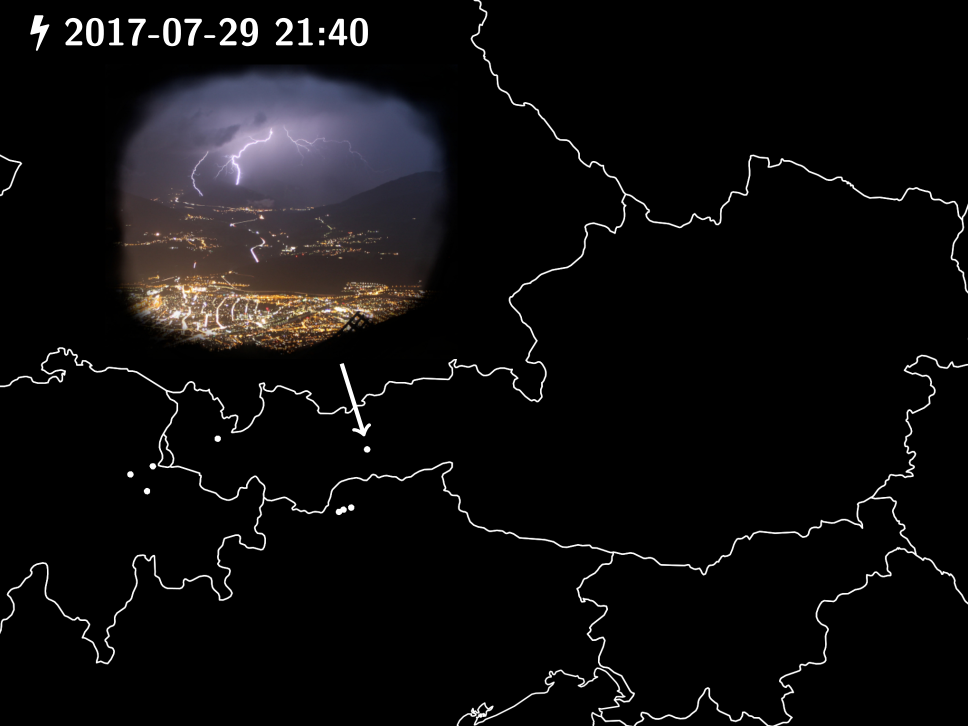
ϟ Spatial Distribution



ϟ Spatial Distribution



⚡ 2017-07-29 21:40



Forecasting Lightning

Model requirements:

- Handle **nonlinear** relationships between the response and covariates.
- **Select** objectively important explanatory variables.
- Provide **inference** of scores and predictions.

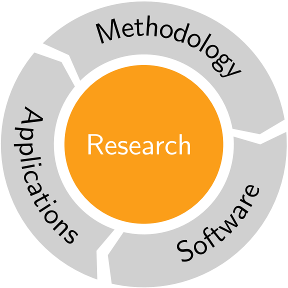
Software requirements:

- Very **flexible** regression model.
- Very **large** dataset.
- Computationally **intensive**.
- Implementation is **not** straightforward.

Lego Toolbox

Hence: Flexible regression framework for Bayesian additive models for location, scale, and shape (BAMLSS).

Software: R package *bamlss*. Modular design supports easy development.



Software Design

Input Data, distribution, regression.

Software Design

Input Data, distribution, regression.

▼

Pre-processing Model frame, transformations.

▼

Software Design

Input Data, distribution, regression.

▼

Pre-processing Model frame, transformations.

▼

Estimation Optimizer and/or sampler functions.

▼

Software Design

Input	Data, distribution, regression.
Pre-processing	Model frame, transformations.
Estimation	Optimizer and/or sampler functions.
Post-processing	Sampling statistics & results.

Software Design

Input	Data, distribution, regression.
Pre-processing	Model frame, transformations.
Estimation	Optimizer and/or sampler functions.
Post-processing	Sampling statistics & results.
Output	Prediction, model selection, visualization, . . .

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

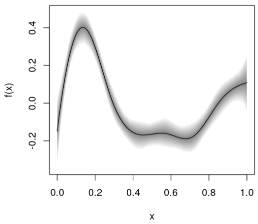
Each parameter is linked to a structured additive predictor

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

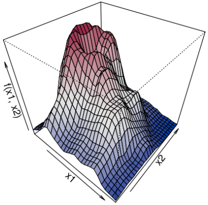
- $j = 1, \ldots, J_k$ and $k = 1, \ldots, K$.
- $h_k(\cdot)$: Link functions for each distribution parameter.
- $f_{jk}(\cdot)$: Model terms of one or more variables.

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

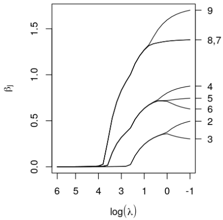
Nonlinear Effects



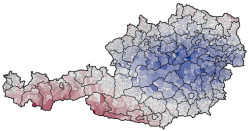
Two-Dimensional Surfaces



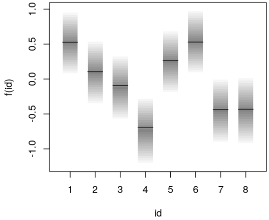
LASSO & Factor Clustering



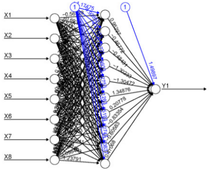
Spatially Correlated Effects f(x) = f(s)



Random Intercepts f(x) = f(id)



Neural Networks



Model Fitting

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

Estimation typically requires to evaluate the log-likelihood

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

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

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.

Priors $p_{jk}(\cdot)$

For simple linear effects $\mathbf{X}_{jk}\boldsymbol{\beta}_{jk}$: $p_{jk}(\boldsymbol{\beta}_{jk}) \propto \mathit{const}$.

For the smooth terms:

$$p_{jk}(\boldsymbol{\beta}_{jk}; \boldsymbol{\tau}_{jk}, \boldsymbol{\alpha}_{jk}) \propto d_{\boldsymbol{\beta}_{jk}}(\boldsymbol{\beta}_{jk} \,|\, \boldsymbol{\tau}_{jk}; \boldsymbol{\alpha}_{\boldsymbol{\beta}_{jk}}) \cdot d_{\boldsymbol{\tau}_{jk}}(\boldsymbol{\tau}_{jk} \,|\, \boldsymbol{\alpha}_{\boldsymbol{\tau}_{jk}}).$$

Using a basis function approach a common choice is

$$d_{\boldsymbol{\beta}_{jk}}(\boldsymbol{\beta}_{jk} \,|\, \boldsymbol{\tau}_{jk}, \boldsymbol{\alpha}_{\boldsymbol{\beta}_{jk}}) \propto |\mathbf{P}_{jk}(\boldsymbol{\tau}_{jk})|^{\frac{1}{2}} \exp\left(-\frac{1}{2}\boldsymbol{\beta}_{jk}^{\top}\mathbf{P}_{jk}(\boldsymbol{\tau}_{jk})\boldsymbol{\beta}_{jk}\right).$$

Precision matrix $\mathbf{P}_{jk}(\boldsymbol{\tau}_{jk})$ derived from prespecified penalty matrices $\boldsymbol{\alpha}_{\boldsymbol{\beta}_{jk}} = \{\mathbf{K}_{1jk}, \ldots, \mathbf{K}_{Ljk}\}$.

The variances parameters $\boldsymbol{\tau}_{jk}$ are equivalent to the inverse smoothing parameters in a frequentist approach.

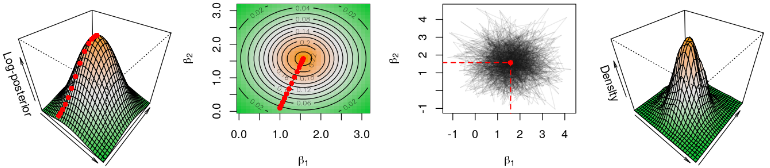
Estimation

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:

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



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 $(\boldsymbol{\beta}^{\star}_{\mathtt{jk}}, \boldsymbol{\tau}^{\star}_{\mathtt{jk}})^{\top} = \mathtt{U}_{\mathtt{jk}}(\boldsymbol{\beta}^{\mathtt{[t]}}_{\mathtt{jk}}, \boldsymbol{\tau}^{\mathtt{[t]}}_{\mathtt{jk}}, \mathbf{y}, \mathbf{X}_{\mathtt{jk}}, \boldsymbol{\alpha}_{jk}, \tilde{\boldsymbol{\eta}})$.
            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.

Updating

Example: MCMC updating functions $\texttt{U}_{jk}(\cdot)$.

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

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

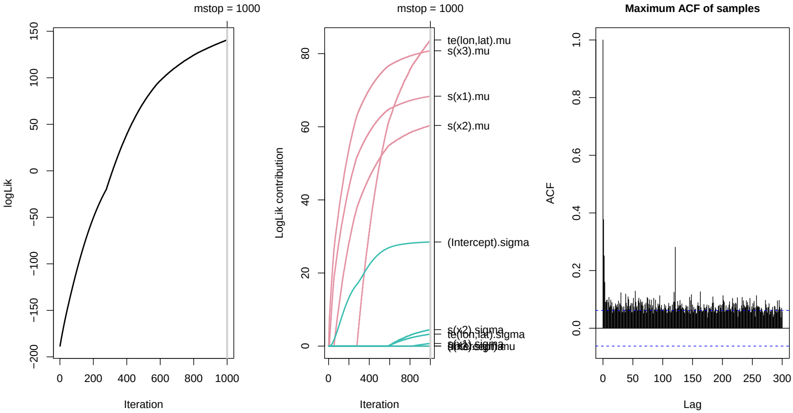
  Metropolis-Hastings acceptance probability

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

- Other sampling schemes, e.g., slice sampling, NUTS, ...

Model Fitting

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



Batchwise Backfitting

Lightning data:

- Lightning dataset includes $>$100 variables from ECMWF ensemble forecasts.
- $\#$Obs. $\sim$ 8M.

Challenges:

- Select only relevant variables.
- Algorithms for very large datasets in distributional regression?
- The aim is to run the analysis for all of Europe!

$\rightarrow$ We need an efficient algorithm with a small memory footprint.

Batchwise Backfitting

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 $\mathbf{u}_k^{[t]}$.

Batchwise Backfitting

SGD type algorithm:

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

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

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!

Batchwise Backfitting

Overfitting:

The idea in batchwise backfitting 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.

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

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

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

Batchwise Backfitting

Convergence:

1. Set, e.g., $\nu = 0.5$, the algorithm will converge after visiting m batches $[\mathbf{s}]$.
2. Optionally, only update if out-of-sample log-likelihood is increased by relative change of ϵ, e.g., $\epsilon = 0.01$.
3. If $\nu = 1$, the algorithm can be interpreted as a bootstrap algorithm and each update $\beta_{jk}^{[t+1]}$ is so to say a "sample". Convergence is achieved similar to MCMC algorithms, i.e., if the iterations start fluctuating around a certain level (here, $\epsilon = -\infty$).
4. Instead of optimizing $\tau_{ljk}^{[t+1]}$, slice sample under $C_{[\tilde{\mathbf{s}}]}(\cdot)$, much faster!

Batchwise Backfitting

Convergence:

Simulated example #Obs. 10000, batches 20, epochs 20.

```
R> set.seed(456)
R> d <- GAMart(10000)
```

Model formula.

```
R> f <- num ~ s(x1,k=20) + s(x2,k=20) + s(x3,k=20)
```

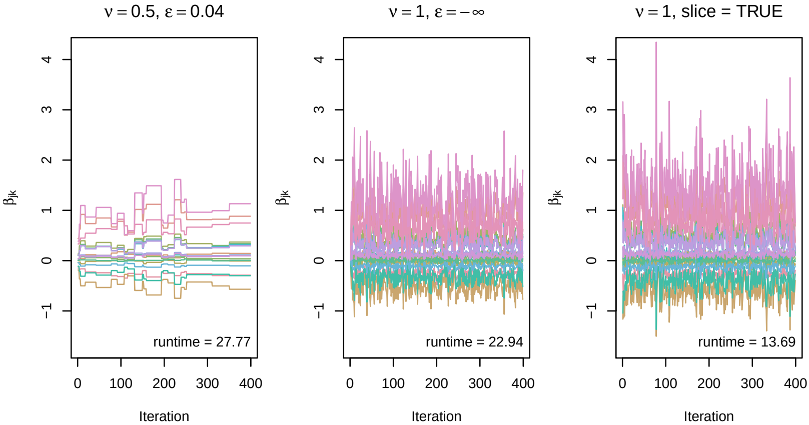
Estimation with batchwise backfitting optimizer function.

```
R> b <- bamlss(f, data = d, sampler = FALSE, optimizer = bbfit,
+     nbatch = 20, epochs = 20, nu = 0.5, aic = TRUE,
+     eps_loglik = 0.04)
```

Batchwise Backfitting

Convergence:

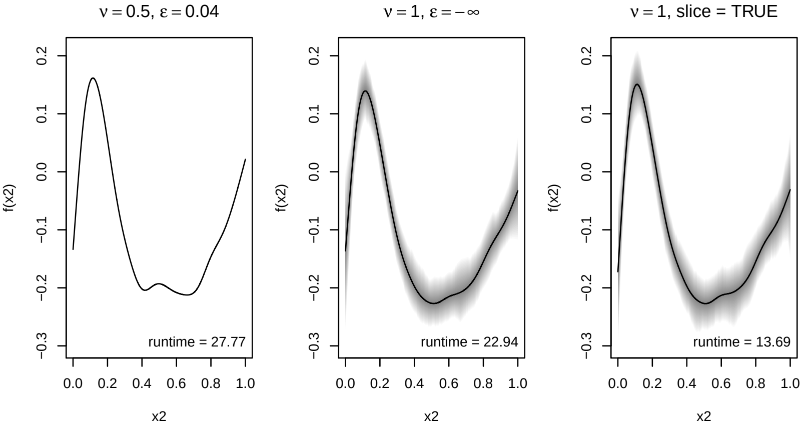
Simulated example #Obs. 10000, batches 20, epochs 20.



Batchwise Backfitting

Convergence:

Simulated example #Obs. 10000, batches 20, epochs 20.



Simulation

We simulated

$$y \sim \mathcal{N}(\mu = \eta_\mu, \log(\sigma) = \eta_\sigma),$$

using

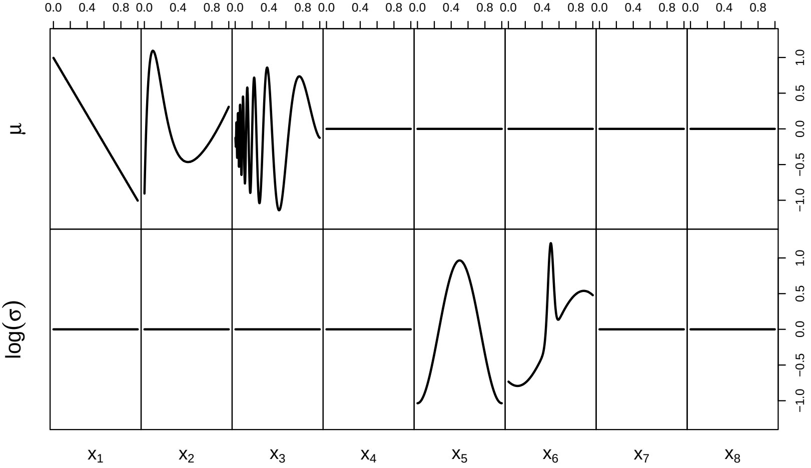
- 100 replications,
- with 1000, 10000 and 50000 observations,
- with $\nu = 0.5$ and $\nu = 0.9$,
- with and without slice sampling,
- compare with *mgcv*

  ```
  gam(..., select = TRUE)
  ```

  and full MCMC in *bamlss*,
- for each model term $f_{jk}(\cdot)$ we use 40 basis functions.

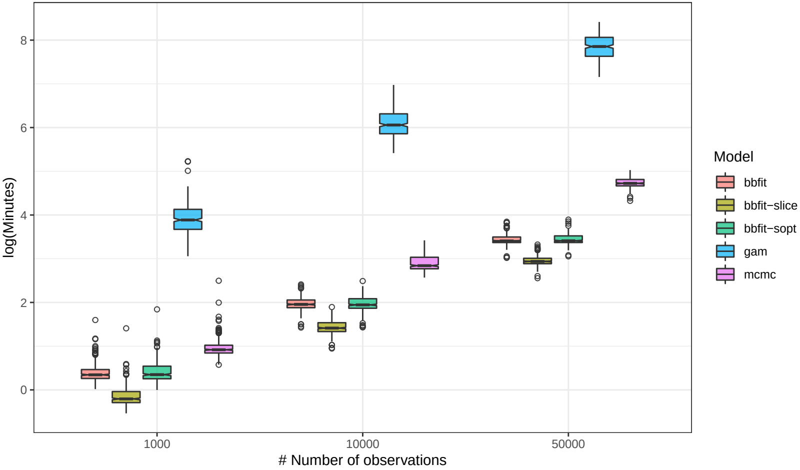
Simulation

Simulated functions:



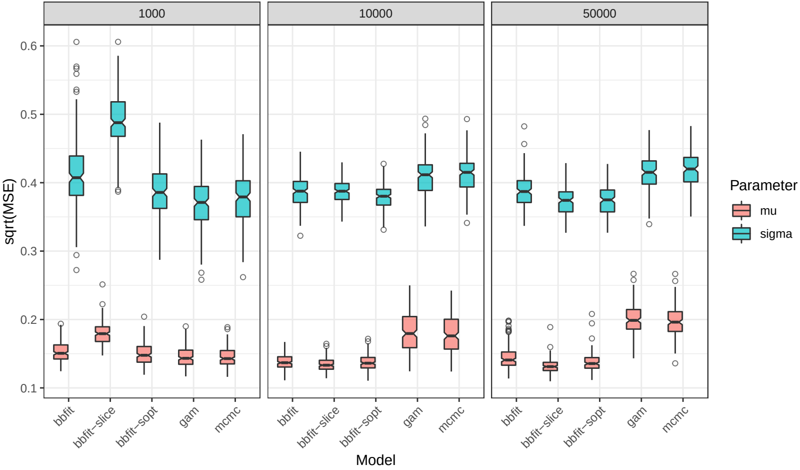
Simulation

Results: Runtimes.



Simulation

Results: MSEs.



Neural Network Terms $f_{jk}(\cdot)$

Motivation:

- Lightning model.
- How to capture complex **nonlinearities** in the atmosphere?
- Neural networks (NN) are **universal** function approximators.

Problems:

- Estimation is difficult and can involve **thousands** of parameters.
- Fully **Bayesian** inference?

Solution:

- Use NNs based on **random** (inner) weights.
- Recently, detailed description on weight sampling available.
- Combine with **LASSO shrinkage**.
- **Batchwise backfitting**, use random weights as starting values, then weights are optimized, e.g., only 20× per $[t]$.

Neural Network Terms $f_{jk}(\cdot)$

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

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

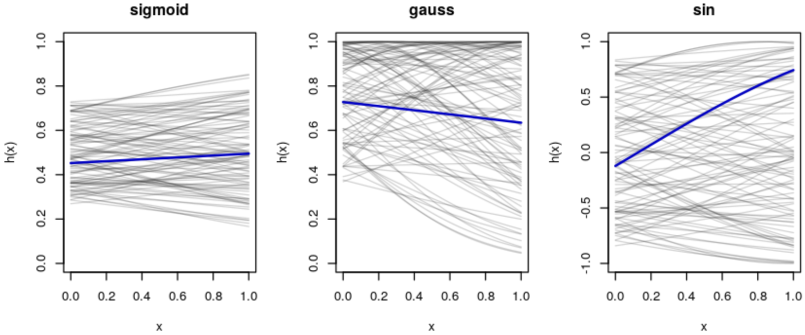
where $\mathbf{w}_l$ and b_l are inner weights and biases.

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

Neural Network Terms $f_{jk}(\cdot)$

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

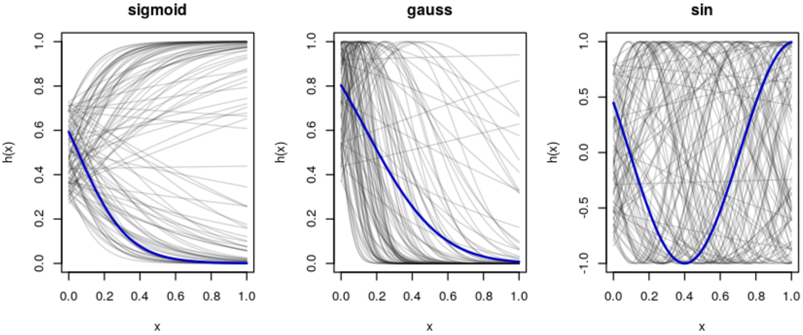
Sample $w_{ld}, b_l \sim \mathcal{U}(-1, 1)$. (Schmidt et al. 1992)



Neural Network Terms $f_{jk}(\cdot)$

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 Terms $f_{jk}(\cdot)$

- 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.

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

Neural Network Terms $f_{jk}(\cdot)$

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

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

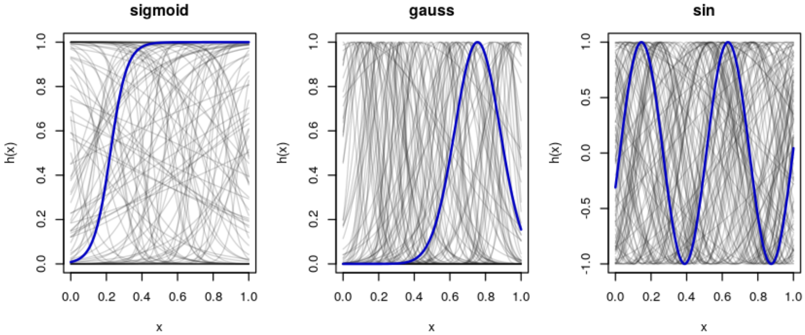
2. For $\mathbf{w}_l$ sample $\zeta_d \sim \mathcal{U}(-1,1)$.
3. Set $w_{ld} = \zeta_d \frac{\sum_{[l]}}{\sum_d \zeta_d}$.
4. Set $b_l = -\sum_d w_{ld} z_l$, where $z_l \sim \mathcal{U}(0,1)$.

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

cc by

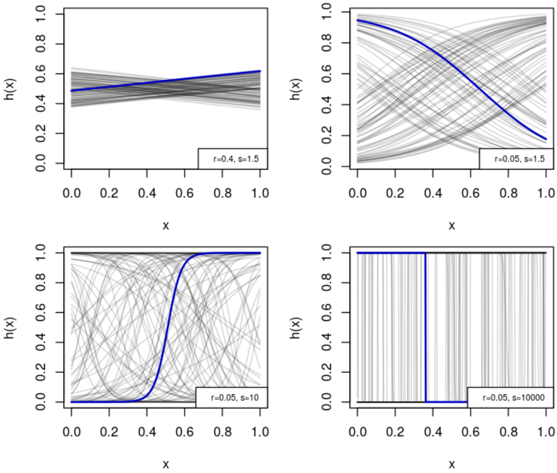
Neural Network Terms $f_{jk}(\cdot)$

Sampling weights: Dudek (2017)



Neural Network Terms $f_{jk}(\cdot)$

Sampling weights: Scaling with r and s.



Elastic net regularization

Overfitting:

We use elastic net regularization

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

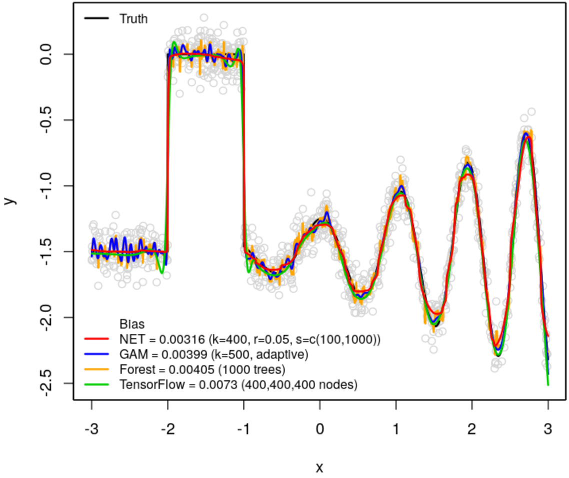
again using quadratic approximations of the LASSO penalties

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

In *bamlss*, this is implemented in the new smooth constructor `n()`.

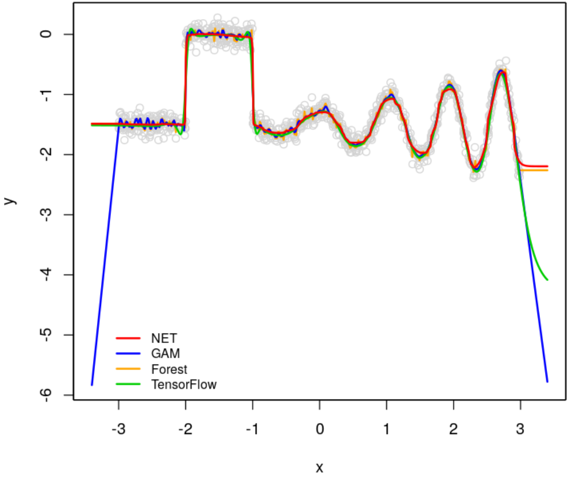
Neural Network Terms $f_{jk}(\cdot)$

Simulated example: Sigmoid activation.



Neural Network Terms $f_{jk}(\cdot)$

Simulated example: Out of range predictions.



Models

Data:
For training we use 2010-2016 data, for testing 2017. We compare with distributional neural network models and *mgcv* `gam()` for the occurrence model.

Regression: Smooth terms for NWP output variables.

Binomial regression:

```
R> f <- flash ~ s(sqrt_cape) + s(d2m) + ...
```

Count regression:

Negative binomial truncated at zero $NB_0(\mu, \theta)$.

```
R> f <- list(
+     counts ~ s(sqrt_cape) + s(d2m) + ...,
+     theta ~ s(sqrt_cape) + s(d2m) + ...
+   )
```

Models

Data:
For training we use 2010-2016 data, for testing 2017. We compare with distributional neural network models and *mgcv* `gam()` for the occurrence model.

Regression: Smooth terms for NWP output variables.

Binomial regression:

```
R> f <- flash ~ sqrt_cape + d2m + ... + n(fn,k=100)
```

Count regression:

Negative binomial truncated at zero $NB_0(\mu, \theta)$.

```
R> f <- list(
+     counts ~ sqrt_cape + d2m + ... + n(fn,k=100),
+     theta ~ sqrt_cape + d2m + ... + n(fn,k=100)
+   )
```

Models

Data:
For training we use 2010-2016 data, for testing 2017. We compare with distributional neural network models and *mgcv* `gam()` for the occurrence model.

Regression: Smooth terms for NWP output variables.

Binomial regression:

```
R> f <- flash ~ s(sqrt_cape) + s(d2m) + ... + n(fn,k=100)
```

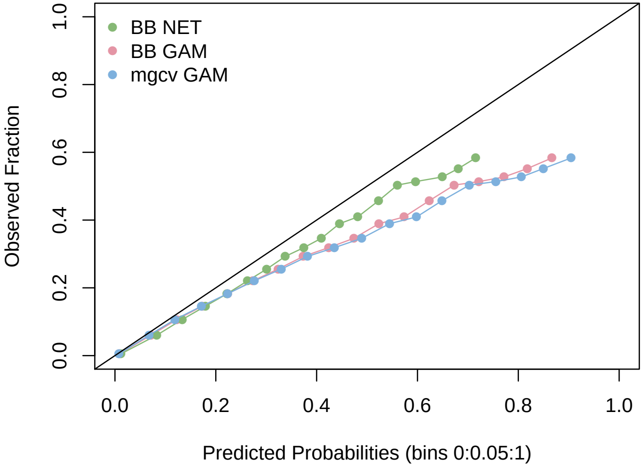
Count regression:

Negative binomial truncated at zero $NB_0(\mu, \theta)$.

```
R> f <- list(
+    counts ~ s(sqrt_cape) + s(d2m) + ... + n(fn,k=100),
+    theta ~ s(sqrt_cape) + s(d2m) + ... + n(fn,k=100)
+  )
```

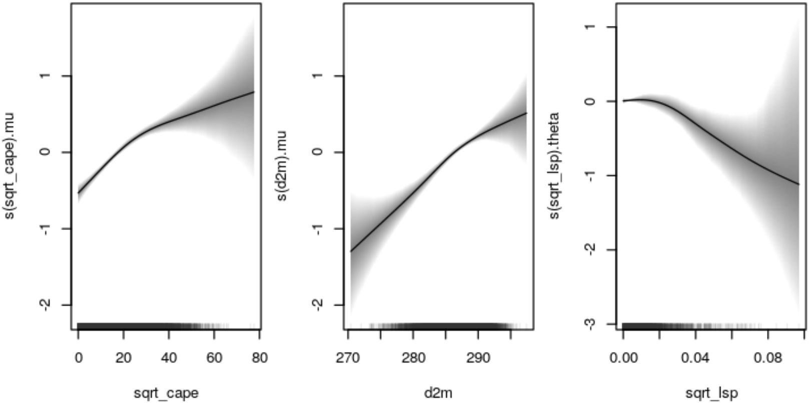
Occurrence Models

Calibration: 2017 out-of-sample data.



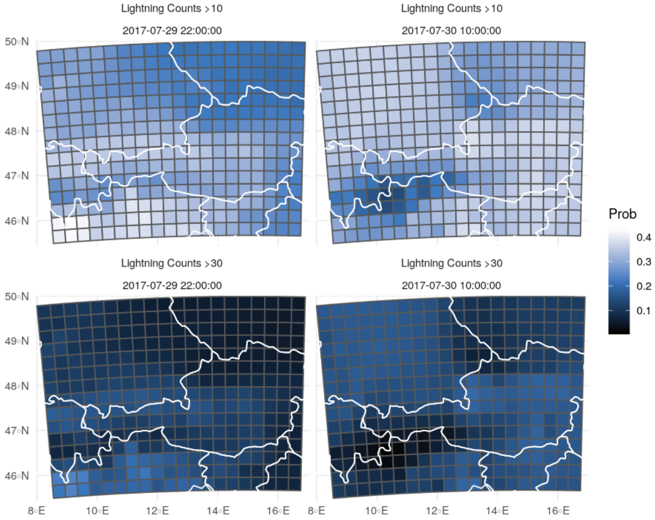
Lightning Models

```
R> plot(b, term = c("s(sqrt_cape)", "s(d2m)", "s(sqrt_lsp)"))
```



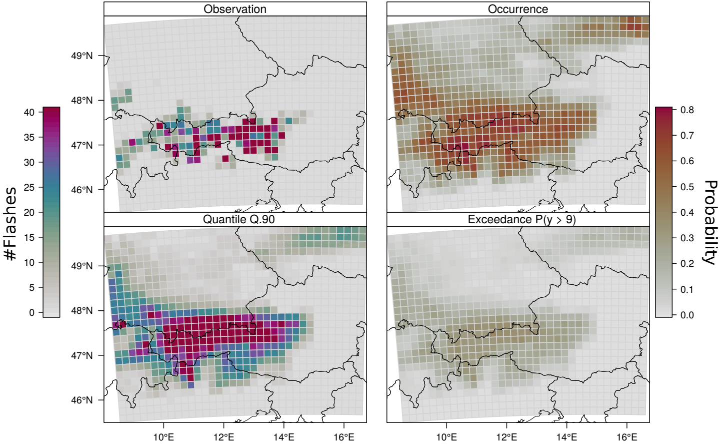
Lightning Models

```
R> p <- predict(b, newdata = nd)
```



NN Lightning Model Predictions

Model performance increase $\sim$ 5% in out-of-sample scores.
(Work in progress)



References

Methodology & Software

- 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`.
- Umlauf et al. (2019). *bamlss: Bayesian Additive Models for Location Scale and Shape (and Beyond)*.
  R package version 1.1-1,
  `http://CRAN.R-project.org/package=bamlss`,
  `http://bamlss.org`.
- 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`.

References

Applications

- Klein, Simon, and Umlauf (2019). *Neural Network Regression with an Application to Leukaemia Survival Data – An Unstructured Distributional Approach*. In Proceedings of the 34th International Workshop on Statistical Modelling, Guimarães, Portugal.
- Simon, Mayr, Umlauf, and Zeileis (2019). *NWP-Based Lightning Prediction Using Flexible Count Data Regression*. Advances in Statistical Climatology, Meteorology and Oceanography, `doi:10.5194/ascmo-5-1-2019`.
- Simon, Fabsic, Mayr, Umlauf, and Zeileis (2018). *Probabilistic Forecasting of Thunderstorms in the Eastern Alps*. Monthly Weather Review, `doi:10.1175/MWR-D-17-0366.1`.