Modular R tools for Bayesian regression are provided by bamlss: From classic MCMC-based GLMs and GAMs to distributional models using the lasso or gradient boosting.

**Abstract**

Over the last decades, the challenges in applied regression and in predictive modeling have been changing considerably: (1) More flexible model specifications are needed as big(ger) data become available, facilitated by more powerful computing infrastructure. (2) Full probabilistic modeling rather than predicting just means or expectations is crucial in many applications. (3) Interest in Bayesian inference has been increasing both as an appealing framework for regularizing or penalizing model estimation as well as a natural alternative to classical frequentist inference. However, while there has been a lot of research in all three areas, also leading to associated software packages, a modular software implementation that allows to easily combine all three aspects has not yet been available. For filling this gap, the R package bamlss is introduced for Bayesian additive models for location, scale, and shape (and beyond). At the core of the package are algorithms for highly-efficient Bayesian estimation and inference that can be applied to generalized additive models (GAMs) or generalized additive models for location, scale, and shape (GAMLSS), also known as distributional regression. However, its building blocks are designed as “Lego bricks” encompassing various distributions (exponential family, Cox, joint models, …), regression terms (linear, splines, random effects, tensor products, spatial fields, …), and estimators (MCMC, backfitting, gradient boosting, lasso, …). It is demonstrated how these can be easily recombined to make classical models more flexible or create new custom models for specific modeling challenges.

**Software**

Replication script: bamlss.R

# ----Packages------------------------------------------------------------

library("bamlss")

library("ggplot2")

library("sf")

## from R-Forge

library("countreg")

# ----Section 2.1: Basic Bayesian regression: Logit model-----------------

data("SwissLabor", package = "AER")

## Model formula

f <- participation ~ income + age + education + youngkids + oldkids + foreign + I(age^2)

## Estimate model.

set.seed(123)

b <- bamlss(f, family = "binomial", data = SwissLabor)

## Model summary.

summary(b)

## Figure 1: MCMC diagnostics.

plot(b, which = c("samples", "max-acf"))

## Predictions on probability scale.

nd <- data.frame(income = 11, age = seq(2, 6.2, length = 100),

education = 12, youngkids = 1, oldkids = 1, foreign = "no")

nd$pSwiss <- predict(b, newdata = nd, type = "parameter", FUN = c95)

nd$foreign <- "yes"

nd$pForeign <- predict(b, newdata = nd, type = "parameter", FUN = c95)

## Plot effect of age on probability.

blues <- function(n, ...) sequential\_hcl(n, "Blues", rev = TRUE)

plot2d(pSwiss ~ age, data = nd, ylab = "participation",

ylim = range(c(nd$pSwiss, nd$pForeign)),

fill.select = c(0, 1, 0, 1))

plot2d(pForeign ~ age, data = nd, add = TRUE,

fill.select = c(0, 1, 0, 1), axes = FALSE,

s2.col = blues, col.lines = blues(1))

legend("topright", c("Foreign", "Swiss"), lwd = 1,

col = c(blues(1), "black"), bty = "n")

# ----Section 2.2: Flexible model terms and estimators--------------------

## Model formula including a smooth model term for age.

f <- participation ~ income + education +

youngkids + oldkids + foreign + s(age, k = 10)

## Estimate model.

set.seed(123)

b <- bamlss(f, family = "binomial", data = SwissLabor)

## Plot estimated smooth effect.

plot(b, term = "s(age)",

ylab = expression(paste("Effect on Logi", t^-1, (pi))))

## Categorize age variable using quantiles.

SwissLabor$cage <- cut(SwissLabor$age,

breaks = quantile(SwissLabor$age, prob = seq(0, 1, length = 10)),

include.lowest = TRUE, ordered\_result = TRUE)

## Model formula including the fused lasso model term for cage.

f <- participation ~ income + education + youngkids + oldkids + foreign + la(cage, fuse = 2)

## Estimate model using the lasso optimizer function.

b <- bamlss(f, family = "binomial", data = SwissLabor,

optimizer = lasso, sampler = FALSE, upper = exp(5), lower = 1,

criterion = "BIC")

## Figure 3: BIC path, paths of coefficients and estimated effect.

par(mfrow = c(1, 3), mar = c(4.1, 4.1, 5, 7.3))

pathplot(b, 1, spar = FALSE)

pathplot(b, 2, spar = FALSE, name = "pi.s.la(cage).cage",

main = "Coefficient paths age", lwd = 2)

page <- predict(b, term = "cage", intercept = FALSE, mstop = lasso\_stop(b))

plot2d(page ~ age, data = SwissLabor, rug = TRUE, lwd = 2,

ylab = expression(paste("Effect on Logi", t^-1, (pi))))

mtext("Estimated\nnonlinear effect", side = 3, line = 1.5, cex = 1.2, font = 2)

# ----Section 2.3: Location-scale model ----------------------------------

data("mcycle", package = "MASS")

## Model formula, one formula for each parameter

## of the distribution.

f <- list(accel ~ s(times, k = 20), sigma ~ s(times, k = 20))

## Estimate model.

set.seed(456)

b <- bamlss(f, data = mcycle, family = "gaussian")

## Visualize estimated effects.

par(mfrow = c(1, 2))

plot(b, model = c("mu", "sigma"))

## Residual diagnostic plots.

par(mfrow = c(1, 2))

plot(b, which = "hist-resid", col = "lightgray", spar = FALSE)

plot(b, which = "qq-resid", spar = FALSE)

## Extract model DIC.

DIC(b)

# ----Section 3: A flexible Bayesian model framework ---------------------

## Model formula from Section 2.1.

f <- participation ~ income + age + education + youngkids + oldkids + foreign + I(age^2)

## Setup the bamlss.frame.

bf <- bamlss.frame(f, data = SwissLabor, family = "binomial")

## Run backfitting model fitting engine.

pm <- with(bf, bfit(x, y, family))

## Run MCMC engine.

set.seed(123)

samps <- with(bf, GMCMC(x, y, family, start = pm$parameters))

## Compute sampling statistics.

stats <- with(bf, samplestats(samps, x, y, family))

print(unlist(stats))

# ----Section 4.1: The BAMLSS model frame --------------------------------

## Generate simulated data set.

set.seed(111)

d <- GAMart()

## A model formula.

f <- list(

num ~ x1 + s(x2) + s(x3) + te(lon,lat),

sigma ~ x1 + s(x2) + s(x3) + te(lon,lat)

)

## Create the bamlss frame.

bf <- bamlss.frame(f, data = d, family = "gaussian")

## Show the structure of the bamlss frame object.

print(bf)

## Show names of smooth term list.

## Each element holds design and penalty matrices

## that can be used for estimation.

print(names(bf$x$mu$smooth.construct))

# ----Section 4.3: Estimation engines ------------------------------------

## Show parameter names of bamlss family.

gaussian\_bamlss()$names

## Example of parameter names.

paste0("mu.s.s(x3)", ".b", 1:10)

# ----Section 5: Flexible count regression for lightning reanalysis ------

## FlashAustria package can be installed from R-Forge

## install.packages("FlashAustria", repos = "http://R-Forge.R-project.org")

## Load the data.

data("FlashAustria", package = "FlashAustria")

## Show some aspects if the data set.

head(FlashAustriaTrain)

nrow(FlashAustriaTrain)

## Specify formula.

f <- list(

counts ~ s(d2m, bs = "ps") + s(q\_prof\_PC1, bs = "ps") +

s(cswc\_prof\_PC4, bs = "ps") + s(t\_prof\_PC1, bs = "ps") +

s(v\_prof\_PC2, bs = "ps") + s(sqrt\_cape, bs = "ps"),

theta ~ s(sqrt\_lsp, bs = "ps")

)

## Load pre-computed model (or re-estimate).

if(TRUE) {

data("FlashAustriaModel", package = "FlashAustria")

b <- FlashAustriaModel

} else {

set.seed(111)

b <- bamlss(f, family = "ztnbinom", data = FlashAustriaTrain,

optimizer = boost, maxit = 1000, ## Boosting arguments.

thin = 5, burnin = 1000, n.iter = 6000) ## Sampler arguments.

}

## Show the loglik contributions of each model term.

pathplot(b, which = "loglik.contrib", intercept = FALSE)

## Show traceplots of MCMC samples.

plot(b, model = "mu", term = "s(sqrt\_cape)", which = "samples")

## Show estimated effects.

par(mfrow = c(1, 3))

plot(b, term = c("s(sqrt\_cape)", "s(q\_prof\_PC1)", "s(sqrt\_lsp)"),

ask = FALSE, spar = FALSE,

rug = TRUE, col.rug = "#39393919")

## Predict parameters of zero-trincated negative binomial model.

fit <- predict(b, newdata = FlashAustriaEval, type = "parameter")

str(fit)

## Show the structure of the bamlss family.

fam <- family(b)

fam

## Compute expected frequencies.

expect <- sapply(1:50, function(j) sum(fam$d(j, fit)))

## Create rootogram to inspect model fit.

names(expect) <- 1:50

expect <- as.table(expect)

obsrvd <- table(FlashAustriaEval$counts)[1:50]

rootogram(obsrvd, expect, xlab = "# Lightning Counts", main = "Rootogram")

## Compute estimated probabilities.

fit <- predict(b, newdata = FlashAustriaCase, type = "parameter")

FlashAustriaCase$P10 <- 1 - fam$p(9, fit)

world <- rnaturalearth::ne\_countries(scale = "medium", returnclass = "sf")

ggplot() + geom\_sf(aes(fill = P10), data = FlashAustriaCase) +

colorspace::scale\_fill\_continuous\_sequential("Oslo", rev = TRUE) +

geom\_sf(data = world, col = "white", fill = NA) +

coord\_sf(xlim = c(7.95, 17), ylim = c(45.45, 50), expand = FALSE) +

facet\_wrap(~time, nrow = 2) + theme\_minimal() +

theme(plot.margin = margin(t = 0, r = 0, b = 0, l = 0))

# ----Appendix A: Custom CRPS() function ---------------------------------

CRPS <- function(object, newdata = NULL) {

yname <- response\_name(object)

fam <- family(object)

if(is.null(fam$p))

stop("no p() function in family object!")

if(is.null(newdata))

newdata <- model.frame(object)

n <- nrow(newdata)

crps <- rep(0, n)

par <- as.data.frame(predict(object, newdata = newdata, type = "parameter"))

for(i in 1:n) {

foo <- function(y) {

(fam$p(y, par[i, , drop = FALSE]) - 1 \* (y >= newdata[[yname]][i]))^2

}

crps[i] <- integrate(foo, -Inf, Inf)$value

}

return(crps)

}

# ----Appendix B: Gaussian family object ---------------------------------

Gauss\_bamlss <- function(...) {

f <- list(

"family" = "mygauss",

"names" = c("mu", "sigma"),

"links" = c(mu = "identity", sigma = "log"),

"d" = function(y, par, log = FALSE) {

dnorm(y, mean = par$mu, sd = par$sigma, log = log)

},

"p" = function(y, par, ...) {

pnorm(y, mean = par$mu, sd = par$sigma, ...)

},

"r" = function(n, par) {

rnorm(n, mean = par$mu, sd = par$sigma)

},

"q" = function(p, par) {

qnorm(p, mean = par$mu, sd = par$sigma)

},

"score" = list(

mu = function(y, par, ...) {

drop((y - par$mu) / (par$sigma^2))

},

sigma = function(y, par, ...) {

drop(-1 + (y - par$mu)^2 / (par$sigma^2))

}

),

"hess" = list(

mu = function(y, par, ...) {

drop(1 / (par$sigma^2))

},

sigma = function(y, par, ...) {

rep(2, length(y))

}

)

)

class(f) <- "family.bamlss"

return(f)

}

# ----Appendix C: Special model terms ------------------------------------

## Smooth construct method.

smooth.construct.gc.smooth.spec <- function(object, data, knots)

{

object$X <- matrix(as.numeric(data[[object$term]]), ncol = 1)

center <- if(!is.null(object$xt$center)) {

object$xt$center

} else TRUE

object$by.done <- TRUE

if(object$by != "NA")

stop("by variables not supported!")

## Begin special elements to be used with bfit() and GMCMC().

object$fit.fun <- function(X, b, ...) {

f <- b[1] \* exp(-b[2] \* exp(-b[3] \* drop(X)))

if(center)

f <- f - mean(f)

f

}

object$update <- bfit\_optim

object$propose <- GMCMC\_slice

object$prior <- function(b) { sum(dnorm(b, sd = 1000, log = TRUE)) }

object$fixed <- TRUE

object$state$parameters <- c("b1" = 0, "b2" = 0.5, "b3" = 0.1)

object$state$fitted.values <- rep(0, length(object$X))

object$state$edf <- 3

object$special.npar <- 3 ## Important!

## End special elements.

## Important, This is a special smooth constructor!

class(object) <- c("gc.smooth", "no.mgcv", "special")

object

}

## Predict matrix method.

Predict.matrix.gc.smooth <- function(object, data, knots)

{

X <- matrix(as.numeric(data[[object$term]]), ncol = 1)

X

}

## Example using the growth curve smooth constructor

## using simulated data.

set.seed(111)

d <- data.frame("time" = 1:30)

d$y <- 2 + 1 / (1 + exp(0.5 \* (15 - d$time))) +

rnorm(30, sd = exp(-3 + 2 \* cos(d$time/30 \* 6 - 3)))

f <- list(

y ~ s2(time, bs = "gc"),

sigma ~ s(time)

)

b <- bamlss(f, data = d, optimizer = bfit, sampler = GMCMC)

## Plot estimated effects.

par(mfrow = c(1, 2))

p <- predict(b, model = "mu", FUN = c95)

plot(y ~ time, data = d, main = expression(mu))

plot2d(p ~ time, data = d, add = TRUE, axes = FALSE,

fill.select = c(0, 1, 0, 1))

points(d$time, d$y)

plot(b, model = "sigma", spar = FALSE, main = expression(log(sigma)))

# ----Appendix C: Model fitting engines for linear regression ------------

## Linear model family object.

lm\_bamlss <- function(...) {

f <- list(

"family" = "LM",

"names" = "mu",

"links" = "identity",

"d" = function(y, par, log = FALSE) {

sigma <- sqrt(sum((y - par$mu)^2) / (length(y) - .lm\_bamlss.p))

dnorm(y, mean = par$mu, sd = sigma, log = log)

},

"p" = function(y, par, ...) {

sigma <- sqrt(sum((y - par$mu)^2) / (length(y) - .lm\_bamlss.p))

pnorm(y, mean = par$mu, sd = sigma, ...)

}

)

class(f) <- "family.bamlss"

return(f)

}

## Simulate some data.

d <- GAMart()

## Test family by creating a bamlss frame.

bf <- bamlss.frame(num ~ x1 + x2, data = d, family = "lm")

print(bf)

## The linear model design matrix.

head(bf$x$mu$model.matrix)

## The response.

head(bf$y)

## Set up an optimizer function that can be used with bamlss().

lm.opt <- function(x, y, ...)

{

## Only univariate response.

y <- y[[1L]]

## For illustration this is easier to read.

X <- x$mu$model.matrix

## Estimate model parameters.

par <- drop(chol2inv(chol(crossprod(X))) %\*% crossprod(X, y))

## Set parameter names.

names(par) <- paste0("mu.p.", colnames(X))

## Return estimated parameters and fitted values.

rval <- list(

"parameters" = par,

"fitted.values" = drop(X %\*% par),

"edf" = length(par),

"sigma" = drop(sqrt(crossprod(y - X %\*% par) / (length(y) - ncol(X))))

)

## Set edf within .GlobalEnv for the

## loglik() function in the lm\_bamlss() family.

.lm\_bamlss.p <<- length(par)

return(rval)

}

## Test the optimizer on the simulated data.

## Note, no MCMC sampling.

f <- num ~ x1 + poly(x2, 5) + poly(x3, 5)

b <- bamlss(f, data = d, family = "lm", optimizer = lm.opt, sampler = FALSE)

summary(b)

nd <- data.frame("x2" = seq(0, 1, length = 100))

nd$p <- predict(b, newdata = nd, term = "x2")

## Plot estimetd effect of x2.

plot2d(p ~ x2, data = nd)

## Set up a sampling engine for linear models.

lm.mcmc <- function(x, y, start = NULL,

n.iter = 12000, burnin = 2000, thin = 10,

m = 0, M = 1e+05,

a = 1, b = 1e-05,

verbose = TRUE, ...)

{

## How many samples are saved?

itrthin <- seq.int(burnin, n.iter, by = thin)

nsaves <- length(itrthin)

## Only univariate response.

y <- y[[1L]]

## For illustration this is easier to read.

X <- x$mu$model.matrix

## Again, set edf within .GlobalEnv for the

## loglik() function in the lm\_bamlss() family.

.lm\_bamlss.p <<- ncol(X)

## Number of observations and parameters.

n <- length(y)

p <- ncol(X)

## Matrix saving the samples.

samples <- matrix(0, nsaves, p + 1L)

## Stick to the naming convention.

pn <- paste0("mu.p.", colnames(X))

colnames(samples) <- c(

pn, ## Regression coefficients and

"sigma" ## variance samples.

)

## Setup coefficient vector,

## again, use correct names.

beta <- rep(0, p)

names(beta) <- pn

sigma <- sd(y)

## Check for starting values obtained,

## e.g., from lm.opt() from above.

if(!is.null(start)) {

sn <- names(start)

for(j in names(beta)) {

if(j %in% sn)

beta[j] <- start[j]

}

}

## Process prior information.

m <- rep(m, length.out = p)

if(length(M) < 2)

M <- rep(M, length.out = p)

if(!is.matrix(M))

M <- diag(M)

Mi <- solve(M)

## Precompute cross products.

XX <- crossprod(X)

Xy <- crossprod(X, y)

## Inverse gamma parameter.

a <- a + n / 2 + p / 2

## Start sampling.

ii <- 1

for(i in 1:n.iter) {

## Sampling sigma

b2 <- b + 1 / 2 \* t(y - X %\*% beta) %\*% (y - X %\*% beta) +

1 / 2 \* t(beta - m) %\*% Mi %\*% (beta - m)

sigma2 <- sqrt(1 / rgamma(1, a, b2))

## Sampling beta.

sigma2i <- 1 / sigma2

Sigma <- chol2inv(chol(sigma2i \* XX + sigma2i \* Mi))

mu <- Sigma %\*% (sigma2i \* Xy + sigma2i \* Mi %\*% m)

beta <- MASS::mvrnorm(1, mu, Sigma)

if(i %in% itrthin) {

samples[ii, pn] <- beta

samples[ii, "sigma"] <- sqrt(sigma2)

ii <- ii + 1

}

if(verbose) {

if(i %% 1000 == 0)

cat("iteration:", i, "\n")

}

}

## Convert to "mcmc" object.

samples <- as.mcmc(samples)

return(samples)

}

## Test both engines on the simulated data.

b <- bamlss(f, data = d, family = "lm", optimizer = lm.opt, sampler = lm.mcmc)

## Show the summary.

summary(b)

## Predict for all terms including 95% credible intervals

nd$x1 <- nd$x3 <- seq(0, 1, length = 100)

for(j in c("x1", "x2", "x3")) {

nd[[paste0("p.", j)]] <- predict(b, newdata = nd, term = j,

FUN = c95, intercept = FALSE)

}

## Plot estimated effects.

par(mfrow = c(1, 3))

plot2d(p.x1 ~ x1, data = nd, fill.select = c(0, 1, 0, 1), lty = c(2, 1, 2))

plot2d(p.x2 ~ x2, data = nd, fill.select = c(0, 1, 0, 1), lty = c(2, 1, 2))

plot2d(p.x3 ~ x3, data = nd, fill.select = c(0, 1, 0, 1), lty = c(2, 1, 2))

**Quick overview**

To illustrate that the bamlss follows the same familiar workflow of the other regression packages such as the basic stats package or the well-established mgcv or gamlss two quick examples are provided: a Bayesian logit model and a location-scale model where both mean and variance of a normal response depend on a smooth term.

The logit model is a basic labor force participation model, a standard application in microeconometrics. Here, the data are loaded from the AER package and the same model formula is specified that would also be used for glm() (as shown on ?SwissLabor).

data("SwissLabor", package = "AER")

f <- participation ~ income + age + education + youngkids + oldkids + foreign + I(age^2)

Then, the model can be estimated with bamlss() using essentially the same look-and-feel as for glm(). The default is to use Markov chain Monte Carlo after obtaining initial parameters via backfitting.

library("bamlss")

set.seed(123)

b <- bamlss(f, family = "binomial", data = SwissLabor)

summary(b)

## Call:

## bamlss(formula = f, family = "binomial", data = SwissLabor)

## ---

## Family: binomial

## Link function: pi = logit

## \*---

## Formula pi:

## ---

## participation ~ income + age + education + youngkids + oldkids +

## foreign + I(age^2)

## -

## Parametric coefficients:

## Mean 2.5% 50% 97.5% parameters

## (Intercept) 6.15503 1.55586 5.99204 11.11051 6.196

## income -1.10565 -1.56986 -1.10784 -0.68652 -1.104

## age 3.45703 2.05897 3.44567 4.79139 3.437

## education 0.03354 -0.02175 0.03284 0.09223 0.033

## youngkids -1.17906 -1.51099 -1.17683 -0.83047 -1.186

## oldkids -0.24122 -0.41231 -0.24099 -0.08054 -0.241

## foreignyes 1.16749 0.76276 1.17035 1.55624 1.168

## I(age^2) -0.48990 -0.65660 -0.49205 -0.31968 -0.488

## alpha 0.87585 0.32301 0.99408 1.00000 NA

## ---

## Sampler summary:

## -

## DIC = 1033.325 logLik = -512.7258 pd = 7.8734

## runtime = 1.417

## ---

## Optimizer summary:

## -

## AICc = 1033.737 converged = 1 edf = 8

## logLik = -508.7851 logPost = -571.3986 nobs = 872

## runtime = 0.012

## ---

The summary is based on the MCMC samples, which suggest “significant” effects for all covariates, except for variable education, since the 95% credible interval contains zero. In addition, the acceptance probabilities alpha are reported and indicate proper behavior of the MCMC algorithm. The column parameters shows respective posterior mode estimates of the regression coefficients, which are calculated by the upstream backfitting algorithm.

To show a more flexible regression model we fit a distributional scale-location model to the well-known simulated motorcycle accident data, provided as mcycle in the MASS package.

Here, the relationship between head acceleration and time after impact is captured by smooth relationships in both mean and variance. See also ?gaulss in the mgcv package for the same type of model estimated with REML rather than MCMC. Here, we load the data, set up a list of two formula with smooth terms (and increased knots k for more flexibility), fit the model almost as usual, and then visualize the fitted terms along with 95% credible intervals.

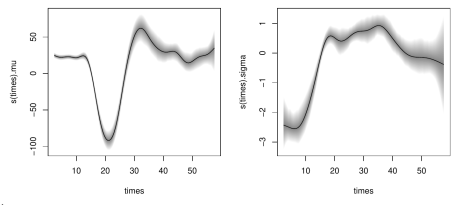
data("mcycle", package = "MASS")

f <- list(accel ~ s(times, k = 20), sigma ~ s(times, k = 20))

set.seed(456)

b <- bamlss(f, data = mcycle, family = "gaussian")

plot(b, model = c("mu", "sigma"))



**Flexible count regression for lightning reanalysis**

Finally, we show a more challenging case study. Here, emphasis is given to the illustration of the workflow. For more details on the background for the data and interpretation of the model, see Section 5 in the full paper linked above. The goal is to establish a probabilistic model linking positive counts of cloud-to-ground lightning discharges in the European Eastern Alps to atmospheric quantities from a reanalysis dataset.

The lightning measurements form the response variable and regressors are taken from the atmospheric quantities from ECMWF’s ERA5 reanalysis data. Both have a temporal resolution of 1 hour for the years 2010-2018 and a spatial mesh size of approximately 32 km. The subset of the data analyzed along with the fitted bamlss model are provided in the FlashAustria data on R-Forge which can be installed by

install.packages("FlashAustria", repos = "<http://R-Forge.R-project.org>")

To model only the lightning counts with at least one lightning discharge we employ a negative binomial count distribution, truncated at zero. The data can be loaded as follows and the regression formula set up:

data("FlashAustria", package = "FlashAustria")

f <- list(

counts ~ s(d2m, bs = "ps") + s(q\_prof\_PC1, bs = "ps") +

s(cswc\_prof\_PC4, bs = "ps") + s(t\_prof\_PC1, bs = "ps") +

s(v\_prof\_PC2, bs = "ps") + s(sqrt\_cape, bs = "ps"),

theta ~ s(sqrt\_lsp, bs = "ps")

)

The expectation mu of the underlying untruncated negative binomial model is modeled by various smooth terms for the atmospheric variables while the overdispersion parameter theta only depends on one smooth regressor. To fit this challenging model, gradient boosting is employed in a first step to obtain initial values for the subsequent MCMC sampler. Running the model takes about 30 minutes on a well-equipped standard PC. In order to move quickly through the example we load the pre-computed model from the FlashAustria package:

data("FlashAustriaModel", package = "FlashAustria")

b <- FlashAustriaModel

But, of course, the model can also be refitted:

set.seed(111)

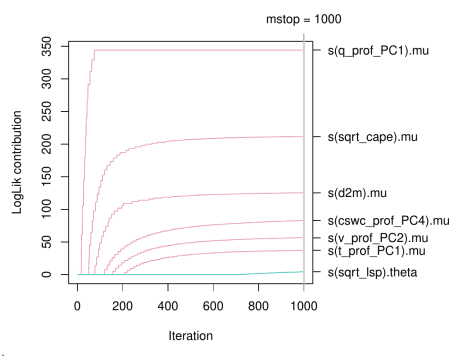
b <- bamlss(f, family = "ztnbinom", data = FlashAustriaTrain,

optimizer = boost, maxit = 1000, ## Boosting arguments.

thin = 5, burnin = 1000, n.iter = 6000) ## Sampler arguments.

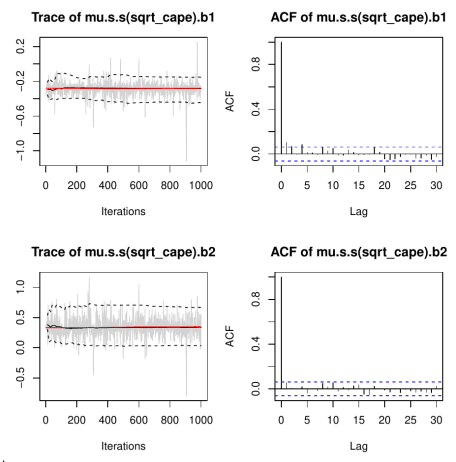
To explore this model in some more detail, we show a couple of visualizations. First, the contribution to the log-likelihood of individual terms during gradient boosting is depicted.

pathplot(b, which = "loglik.contrib", intercept = FALSE)



Subsequently, we show traceplots of the MCMC samples (left) along with autocorrelation for two splines the term s(sqrt\_cape) of the model for mu.

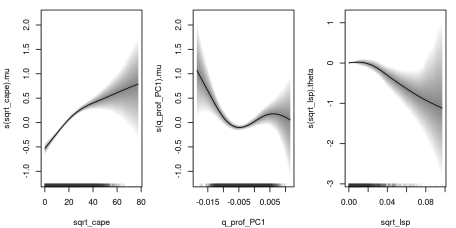
plot(b, model = "mu", term = "s(sqrt\_cape)", which = "samples")



Next, the effects of the terms s(sqrt\_cape) and s(q\_prof\_PC1) from the model for mu and term s(sqrt\_lsp) from the model for theta are shown along with 95% credible intervals derived from the MCMC samples.

plot(b, term = c("s(sqrt\_cape)", "s(q\_prof\_PC1)", "s(sqrt\_lsp)"),

rug = TRUE, col.rug = "#39393919")



Finally, estimated probabilities for observing 10 or more lightning counts (within one grid box) are computed and visualized. The reconstructions for four time points on September 15-16, 2001 are shown.

fit <- predict(b, newdata = FlashAustriaCase, type = "parameter")

fam <- family(b)

FlashAustriaCase$P10 <- 1 - fam$p(9, fit)

world <- rnaturalearth::ne\_countries(scale = "medium", returnclass = "sf")

library("ggplot2")

ggplot() + geom\_sf(aes(fill = P10), data = FlashAustriaCase) +

colorspace::scale\_fill\_continuous\_sequential("Oslo", rev = TRUE) +

geom\_sf(data = world, col = "white", fill = NA) +

coord\_sf(xlim = c(7.95, 17), ylim = c(45.45, 50), expand = FALSE) +

facet\_wrap(~time, nrow = 2) + theme\_minimal() +

theme(plot.margin = margin(t = 0, r = 0, b = 0, l = 0))

