# Introduction to gradient boosting for extremes

*Clément Dombry and Jasper Velthoen*

*10/18/2019*

## Extreme quantile regression via GPD modelling of exceedances above high threshold and

By the Pickands-de Haan-Balkema theorem, if the rescaled distribution of exceedances converges to a non degenerate distribution

$$\lim_{u\uparrow y^*} \mathbb{P}\left(\frac{Y-u}{f(u)} > y \,|\, Y > u\right) = 1 - H(y)$$

the limit is necessarily a GPD distribution

$$H_{\gamma,\sigma}(y) = 1 - \left(1+\gamma\frac{y}{\sigma}\right)_+^{-1/\gamma}.$$

This happens if and only if $Y$ is in the max-domain of attraction of the extreme value distribution with index $\gamma$. The probability to exceed the high threshold $u$ is denoted by $p = \mathbb{P}(Y > u)$. The approximation

$$\mathcal{L}(Y - u \,|\, Y > u) \overset{d}{\approx} H_{\gamma,\tilde{\sigma}}, \quad \tilde{\sigma} = f(u)\sigma,$$

implies that the quantiles of high order $1-\alpha$ are approximated by

$$q(\alpha) \approx u + \tilde{\sigma}\frac{(\alpha/p)^{-\gamma}-1}{\gamma}, \quad \alpha < p.$$

For a covariate dependent model, the high threshold $u(x)$ may depend on $x$ and we assume for simplicity that the exceedances are exactly GPD. This yields the model

$$\begin{cases} p(x) = \mathbb{P}(Y > u(x) \,|\, X = x), \\ \mathcal{L}(Y - u(x) \,|\, Y > u(x), X = x) = H_{\gamma(x),\sigma(x)}. \end{cases}$$

The corresponding conditional quantile for $Y$ given $X = x$ with order $1-\alpha$ is

$$q(\alpha|x) = u(x) + \sigma(x)\frac{(\alpha/p(x))^{-\gamma(x)}-1}{\gamma(x)}, \quad \alpha < p(x).$$

The estimation of $q(\alpha|x)$ is then obtained by plugging estimations of $p(x), \sigma(x)$ and $\gamma(x)$.

For GPD estimation, maximum likelihood estimation is standard and provides asymptotically normal estimators in the i.i.d. case with $\gamma > -1/2$. It is natural to introduce the rescaled exceedance $Z = \max(Y - u(x), 0)$, where $Z = 0$ corresponds to no exceedance. The negative log-likelihood at $\theta = (p, \sigma, \gamma)$ writes

$$\begin{aligned} \ell_z(\theta) =& \log(1-p))1_{\{z=0\}} - \log p 1_{\{z>0\}} \\ &+ \left[(1+1/\gamma)\log\left(1+\gamma\frac{z}{\sigma}\right) + \log\sigma\right]1_{z>0}. \end{aligned}$$

The first line corresponds to the Bernoulli negative log-likelihood and is related to the classification problem ($z > 0$) VS ($z = 0$), that is exceedance VS no exceedance. The second line is the GPD negative log-likelihood and is related to the GP modeling for exceedances $z > 0$ only.

In a statistical learning framework, we need to learn the function

$$\theta(x) = (p(x), \sigma(x), \gamma(x))$$

from an i.i.d. sample of observations $(x_i, z_i)$, $1 \leq i \leq n$. For simplicity, we assume a deterministic threshold $u(x)$. Our proposal is to use gradient boosting (Friedman) and generalized random forest (Athey, Wager, Tisbshirani), mostly from a methodological point of view because theoretical properties seem difficult to tackle. In the following, we consider gradient boosting that is more straightforward to write in our specific setting and easier to code.

## Gradient boosting for extreme quantile regression

We refer to The Elements of Statistical Learning, section 10.10 (Numerical Optimization via Gradient Boosting). The following algorithm is a natural adaptation of Algorithm 10.3 therein. The main difference is that in our framework, the function to learn $\theta(x)$ has three components, so that we will learn three sequences of trees - a similar strategy is used in multiclass classification where several sequences of trees are trained to learn the probabilities of the different classes (see Algorithm 10.4 in ESL).

For $\theta = (p, \sigma, \gamma)$, we use the notation $\theta = (\theta^p, \theta^\sigma, \theta^\gamma)$ and one generic component is noted $\theta^\delta$ with $\delta \in \{p, \sigma, \gamma\}$ . The algorithm runs as follows:

- data set: $(x_i, z_i)$, $1 \leq i \leq n$.
- parameters:
  - $B$ number of trees (same for three sequences),
  - $\lambda = (\lambda^p, \lambda^\sigma, \lambda^\gamma)$ learning rates,
  - $J = (J^p, J^\sigma, J^\gamma)$ numbers of leaves in the trees.
- Procedure:
  1. Initialize at
  $$\theta_0(x) \equiv \arg\min_\theta \sum_{i=1}^n \ell_{z_i}(\theta).$$
  This is simply the maximum likelihood estimator when the $x_i$'s are ignored and the $z_i$'s assumed i.i.d. with likelihood $\ell$.
  2. For $b = 1$ to $B$:
     (a) (gradient) For $\delta \in \{p, \sigma, \gamma\}$, compute the partial derivatives
     $$r_{bi}^\delta = \frac{\partial \ell_{z_i}}{\partial \theta^\delta}(\theta_{b-1}(x_i)), \quad 1 \leq i \leq n.$$
     (b) (tree regions) For $\delta \in \{p, \sigma, \gamma\}$, fit a regression tree $r_{bi}^\delta \sim x_i$, yielding $J^\delta$ terminal regions $R_{bj}^\delta$, $1 \leq j \leq J^\delta$.
     (c) (tree values with line search) For $\delta \in \{p, \sigma, \gamma\}$ and $1 \leq j \leq J^\delta$, compute
     $$\gamma_{bj}^\delta = \arg\min_\gamma \sum_{x_i \in R_{bj}^\delta} \ell_{z_i}(\theta_{b-1}(x_i) + \gamma).$$
     where $\gamma$ acts on the $\delta$-component only; define the tree
     $$T_b^\delta(x) = \sum_{j=1}^{J^\delta} \gamma_{bj}^\delta 1_{\{x \in R_{bj}^\delta\}}.$$

(d) (update) For $\delta \in \{p, \sigma, \gamma\}$, update

$$\theta_b^\delta(x) = \theta_{b-1}^\delta(x) + \lambda^\delta T_b^\delta(x).$$

- Output: $\hat{\theta}(x) = \theta_B(x)$. The $\delta$-component is the sum

$$\hat{\theta}(x) = \theta_0^\delta + \lambda^\delta \sum_{b=1}^{B} T_b^\delta(x).$$

## Implementation in the gbex package

The current implementation in the gbex we assume that the data follow a GPD distribution with parameters some function $\sigma(x)$ and $\gamma(x)$. First we can install and load the package,

```
require(rpart)
```

```
## Loading required package: rpart
```

```
require(treeClust)
```

```
## Loading required package: treeClust
```

```
## Loading required package: cluster
```

```
require(magrittr)
```

```
## Loading required package: magrittr
```

```
require(dplyr)
```

```
## Loading required package: dplyr
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
package_directory <- "/Users/jjvelthoen/Dropbox/PhD/gradient boosting extremes/gbex"
install.packages(package_directory,repos=NULL,type="source",quiet=T)
library(gbex)
```

First we need data, which will be simulated from the following model,

$$\begin{aligned}
&X_1, X_2 \sim \text{Unif}(-1, 1) \\
&\overline{X} = X_1^2 + X_2^2 \\
&\sigma(\overline{X}) = \exp(\overline{X}) \\
&\gamma(\overline{X}) = \frac{1}{3} + \frac{1}{10}\overline{X} \\
&Y \sim GPD(\sigma(\overline{X}), \gamma\overline{X})
\end{aligned}$$

This can be done with a function in the package,

```
set.seed(25051979)

### Data generating process ###
n <- 1000
d <- 2
data <- get_data(n,d)
s <- data$s # the sigma parameter
g <- data$g # the gamma parameter
y <- data$y # the response vector
print("The response vector:")
```

```
## [1] "The response vector:"
```

```
head(y)
```

```
## [1] 0.7188297 0.8205733 0.2968209 4.3671856 2.5397324 6.0662247
```

```
X <- data.frame(X=data$X) # the covariates
print("The covariate matrix:")
```

```
## [1] "The covariate matrix:"
```

```
head(X)
```

```
##          X.1        X.2
## 1 -0.5349841  0.3632613
## 2 -0.4678420 -0.5152282
## 3 -0.1079186 -0.8660849
## 4 -0.3912828 -0.3083739
## 5  0.9459987  0.2457791
## 6  0.5998966 -0.4821159
```

Before applying the gradient boosting several tuning parameters need to be specified,

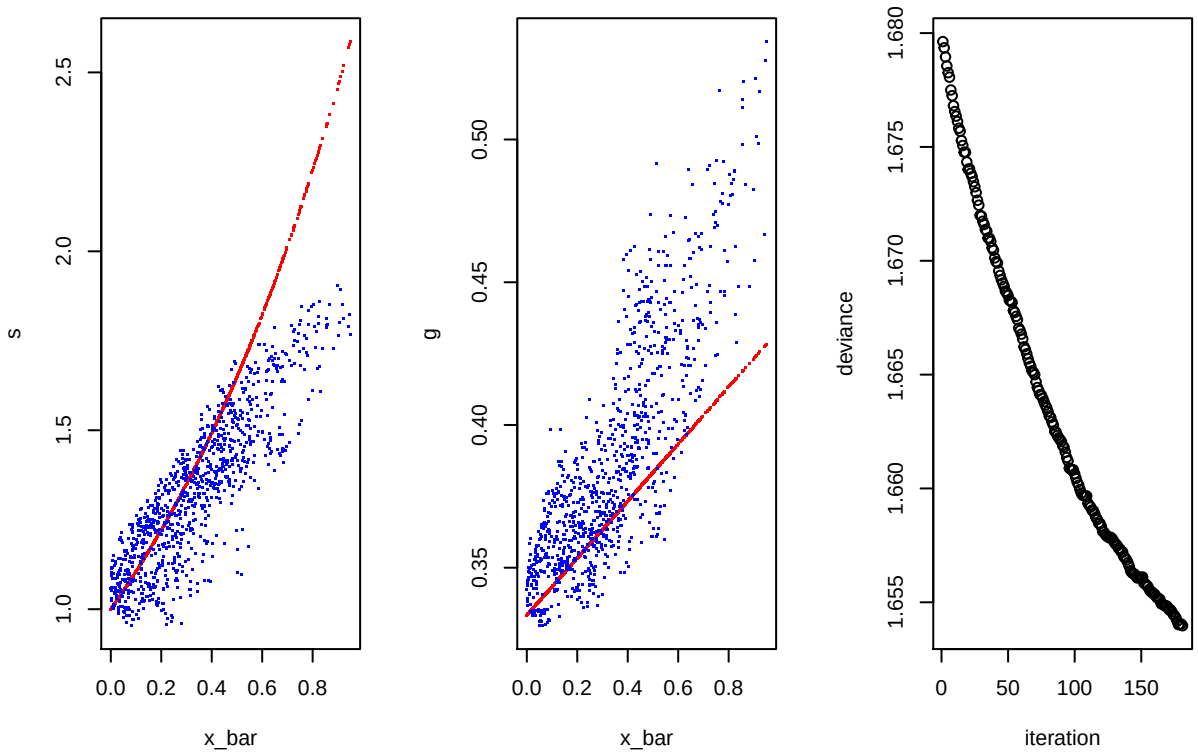
```
B <- 180  # The number of gradient boosting steps
lambda=c(0.025,0.0025) # The learning rate for the sigma and gamma parameter
depth=c(2,2) # The maximum depth for a single sigma and gamma tree
min_leaf_size=c(4,4) # The minimum number of observations contained in the leafnodes
sf=0.1 # The subsample fraction used for estimating each tree (in a single step only one subsample is d
```

The model is now fitted using the *gbex* function,

```
fit <- gbex(y,X,B=B,lambda=lambda,depth=depth,min_leaf_size=min_leaf_size,sf=sf)
```

To see how the fit performs some diagnostics can be plotted where in red the true parameter and in blue the estimated ones.

```
x_bar=apply(X^2, 1,mean)
par(mfrow=c(1,3))
plot(x_bar,s,pch='.', col='red',ylim=c(min(c(s,fit$s_hat)),max(c(s,fit$s_hat))))
points(x_bar,fit$s_hat,pch='.',col='blue')
plot(x_bar,g,pch='.', col='red',ylim=c(min(c(g,fit$g_hat)),max(c(g,fit$g_hat))))
points(x_bar,fit$g_hat,pch='.',col='blue')
plot(fit$dev,xlab="iteration",ylab="deviance")
```



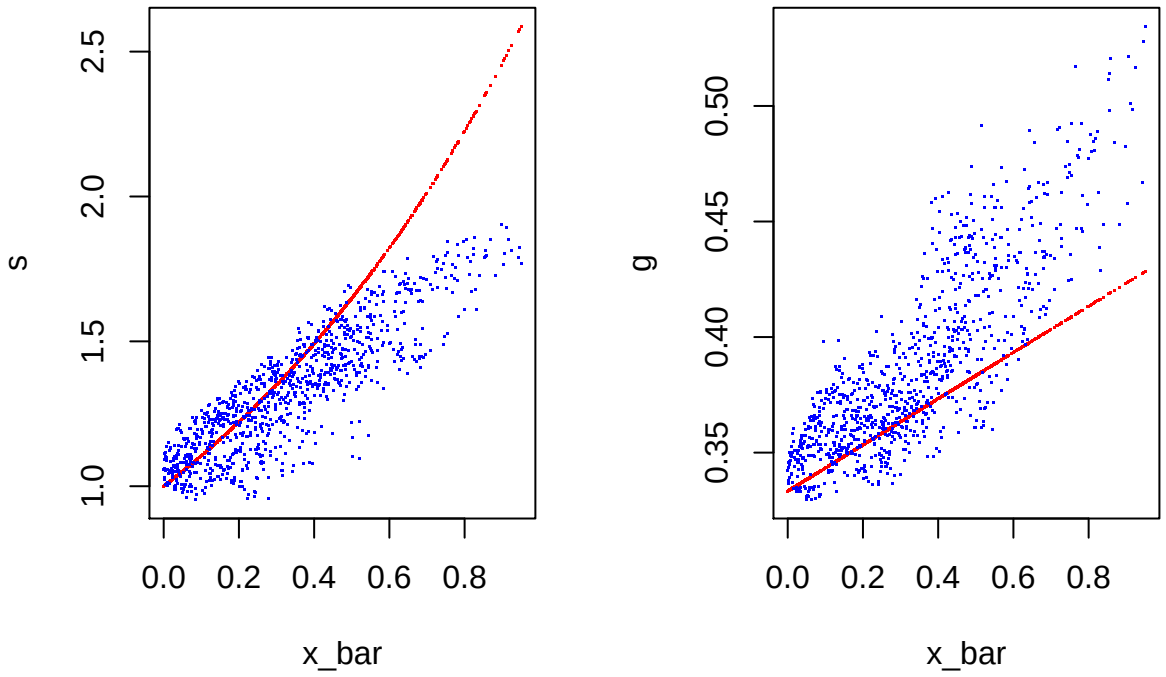

The fit is quite well for the lower values of $\overline{X}$, but for high values the $\sigma$ parameter is harder to estimate because of boundary bias which leads to an opposite bias in the $\gamma$.

The gbex package also has a predict function that can be used to use the fitted gbex object to predict.

```
y_test <- s*(runif(n)^(-g)-1)/g
theta_hat <- predict(fit,newdata=X)
s_hat_test <- theta_hat$s
g_hat_test <- theta_hat$g
```

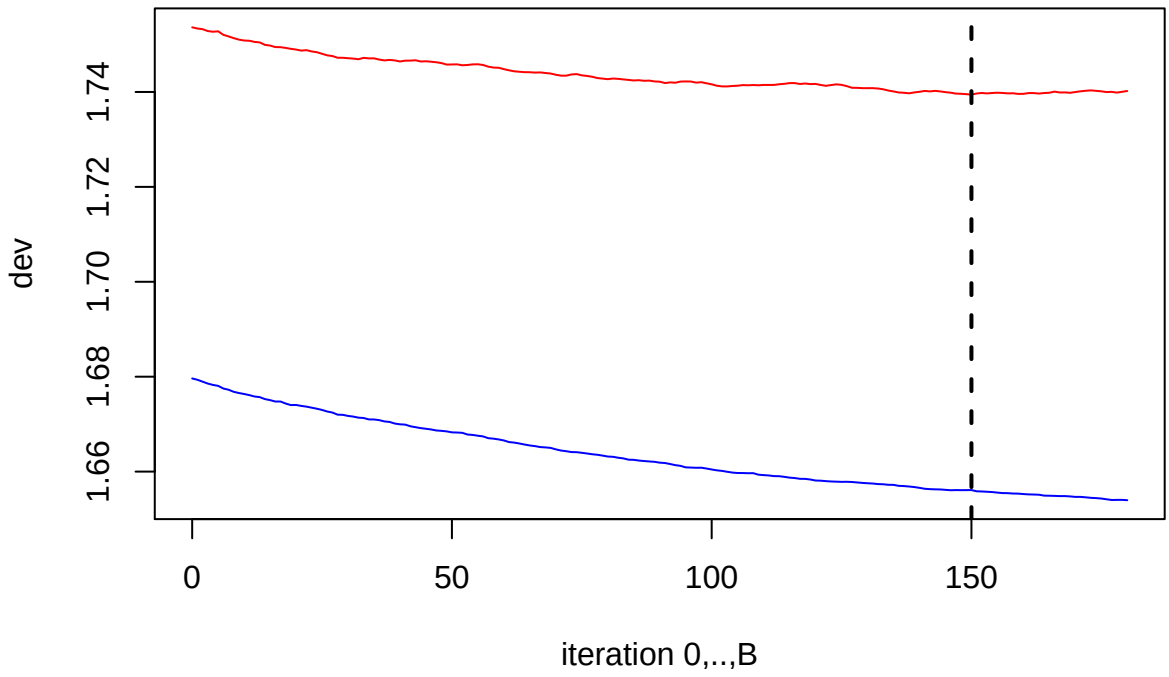
visualizing these new estimates gives us the following

```
par(mfrow=c(1,2))
plot(x_bar,s,pch='.', col='red',ylim=c(min(c(s,s_hat_test)),max(c(s,s_hat_test))))
points(x_bar,s_hat_test,pch='.',col='blue')
plot(x_bar,g,pch='.', col='red',ylim=c(min(c(g,g_hat_test)),max(c(g,g_hat_test))))
points(x_bar,g_hat_test,pch='.',col='blue')
```




Important is to check that gbex is not overfitting by using too many iteration by investigating the deviance on the test_set

```
dev <- dev_per_step(fit)
dev_test <- dev_per_step(fit,X=X,y=y_test)
minimum_iteration <- (0:B)[which(dev_test==min(dev_test))]
plot(0:B,dev,type="l",xlab="iteration 0,..,B",col="blue",ylim=c(min(dev,dev_test),max(dev,dev_test)))
lines(0:B,dev_test,col="red")
abline(v=minimum_iteration,lty=2,lwd=2)
```



## Some notes on the implementation of gbex

The current version of the implementation does not contain the linesearch, but instead the average derivative in each leafnode is used. In order to make the predict function we need tree objects defined as $T_b^\delta(x)$. Though the tree objects from rpart contain the derivatives of the negative likelihood. Instead of manually changing the rpart object our trees will be a list with the rpart object together with a data.frame with the new values for each leaf node, i.e. the $\gamma_{bj}^\delta$. Then in order to predict we can use the package *treeClust* to obtain for a new set of observations the corresponding leaf nodes.

The amount of tuning parameters that has to be chosen is quite large. It is important to note that the tuning parameters sometimes tune the same thing. In my opinion the subsampling fraction is one of the most important. This allows to select a small number of observations making the procedure not too sensitive in the presence of very high observations $Y$. From little simulation experience I also see that correctly specifying the learning rate gives considerable gain.