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 γ. 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 θ^δ 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 ℓ.
  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^δ terminal regions R_{bj}^δ, $1 \leq j \leq J^\delta$.
     (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 γ acts on the δ-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 δ-component is the sum

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

The gbex function

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,

```
rm(list=ls())
require(rpart)
```

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

```
require(treeClust)
```

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

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

```
package_directory <- "/Users/jjvelthoen/Documents/GitHub/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,

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

The data can be generated with the *get_data* function in the package,

```
set.seed(1234)

### 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$y # the response vector
X <- data.frame(X=data$X) # the covariates
colnames(X) = paste0("X",1:ncol(X))
```

The model contains several tuning parameters which can be tuned later by cross validation, for an inital fit we keep them fixed.

```
B <- 250  # Number of trees
lambda=c(0.01,0.0025) # Learning rate for sigma and gamma
depth=c(2,2) # maximum depth of trees for sigma and gamma
min_leaf_size=c(10,10) # minimum_leaf_size of trees for sigma and gamma
sf=0.75 # Subsampling fraction to use for each tree
```

Now the model is fitted using the *gbex* function.

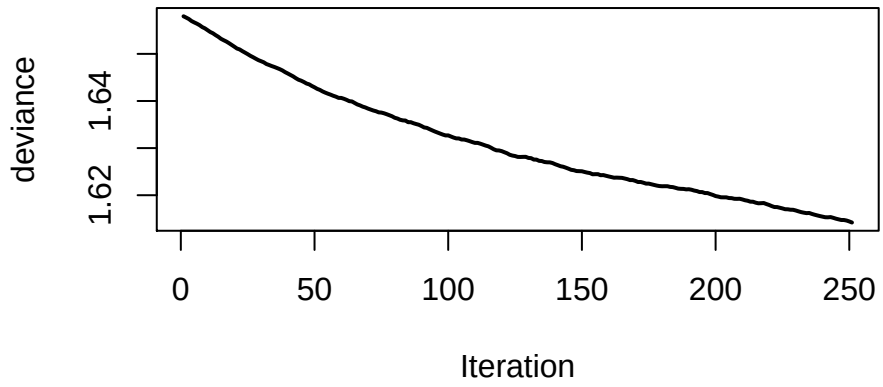
```
fit <- gbex(y,X,B=B,lambda=lambda,depth=depth,min_leaf_size=min_leaf_size,sf=sf,alpha=0,silent=T)
print(fit)
```

```
## gbex(y = y, X = X, B = B, lambda = lambda, depth = depth, min_leaf_size = min_leaf_size,
##      sf = sf, alpha = 0, silent = T)
## A gradient boosting model for extremes fitted by likelihood.
## 250 trees are fitted.
## Training error was equal to 1.6142066602263.
```

The model is fitted by gradient descent using maximum likelihood, alpha = 0, for different parameters power divergence is used with power alpha.

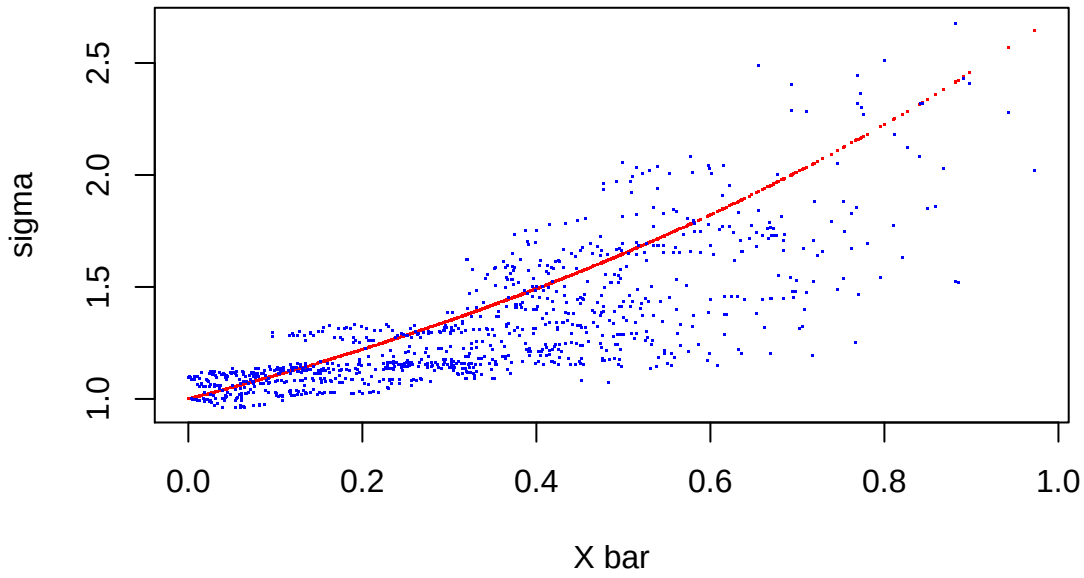
Now we can investigate how the model fit is on the training data and see if it indeed minimizes the negative log-likelihood.

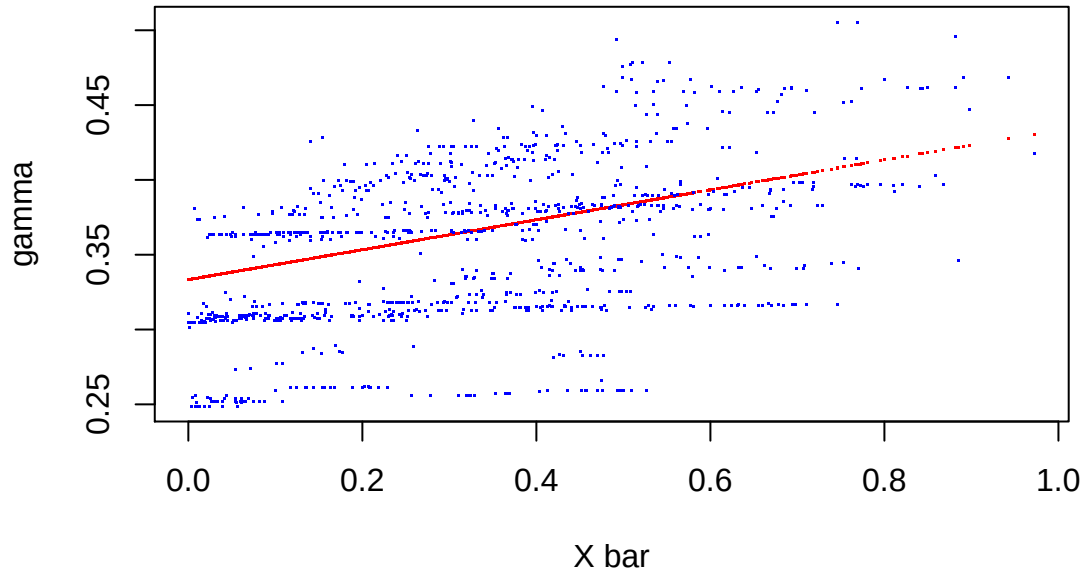
```
plot(fit$dev,type="l",lwd=2,xlab="Iteration",ylab="deviance")
```



Let's see how the parameters depend on the covariates. In the below plot in blue are the estimated parameters and in red the true parameters plotted against $\overline{X}$.

```
x_bar=apply(X^2, 1,mean)
par(mfrow=c(2,1))
plot(x_bar,s,pch='.', col='red',ylim=c(range(c(s,exp(fit$theta$b)))),
     xlab="X bar",ylab="sigma")
points(x_bar,exp(fit$theta$b),pch='.',col='blue')
plot(x_bar,g,pch='.', col='red',ylim=c(range(c(g,fit$theta$g))),
     xlab="X bar",ylab="gamma")
points(x_bar,fit$theta$g,pch='.',col='blue')
```

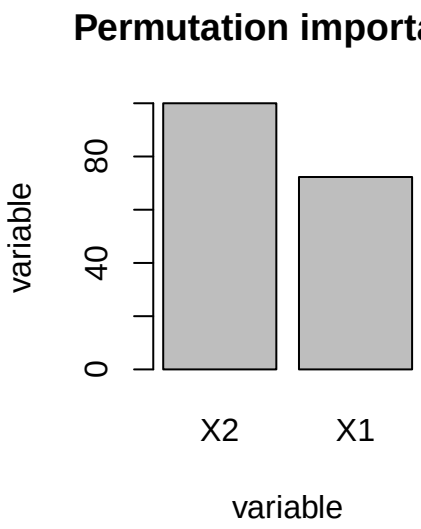




In practice we do not have the true parameters, hence we can not make the above plot. Though in order to inspect the model fit several graphs can be made. First we inspect what the contribution is of the different covariates.

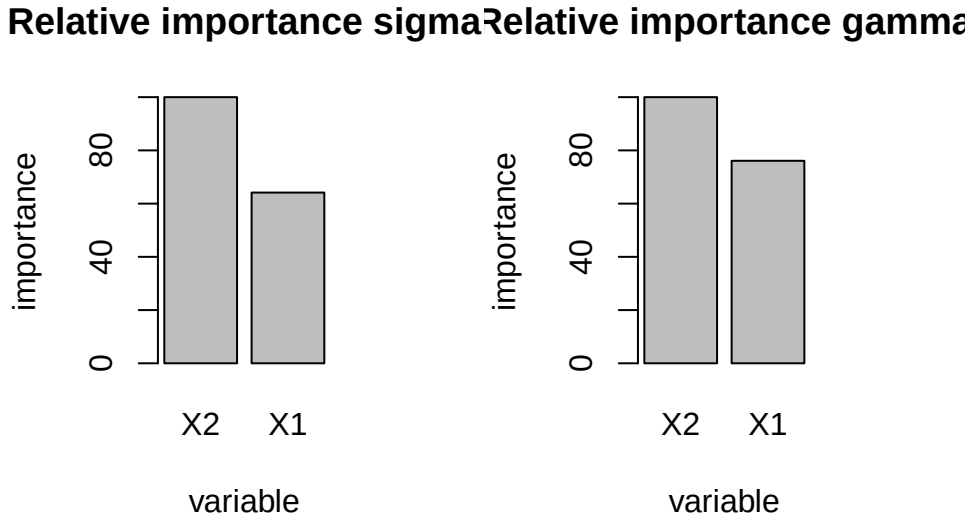
For this we look at the permutation variable importance, which measures the increase in deviance when all values for a single covariate are permuted. These values for each covariate is then rescaled such that the largest is equal to 100.

```
variable_importance(fit,type="permutation")
```



Additionally, instead of permutation variable importance we can inspect the relative importance of the variables for estimating the two variables.

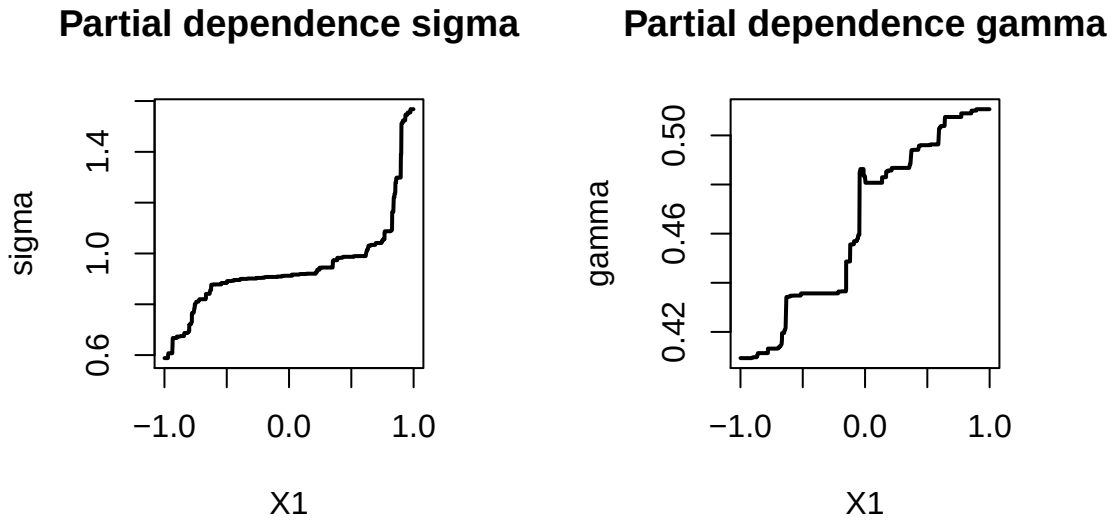
```
variable_importance(fit,type="relative")
```



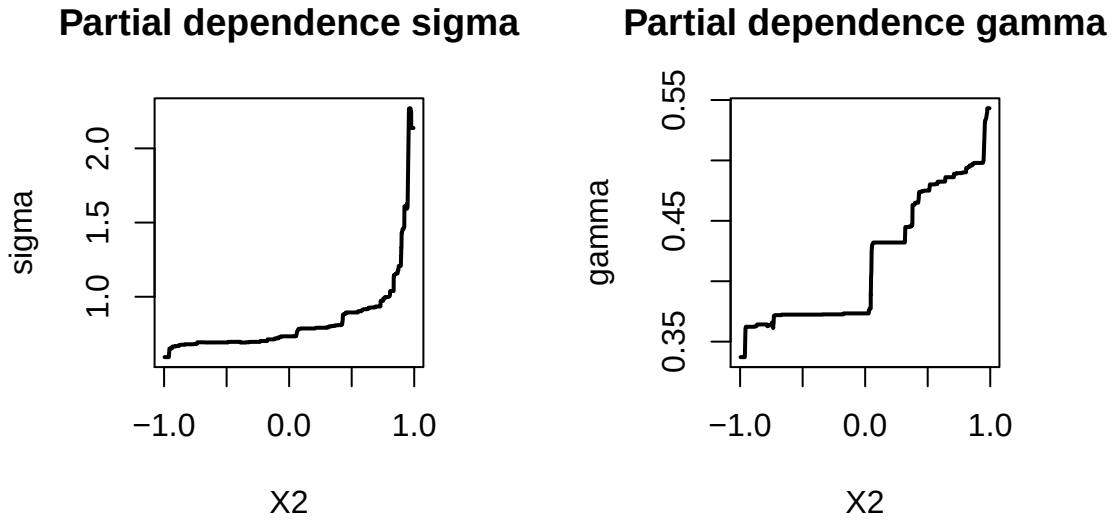
Although the model is symmetric in X_1 and X_2 the X_2 variable appears to be more important for this set of realizations.

Finally, it would be of interest to investigate the dependence of the covariates on the parameters σ and γ. Hereto a partial dependence plot gives some information about this.

```
partial_dependence(fit,variable = 1)
```




```
partial_dependence(fit,variable = 2)
```



Cross validation with the gbex function

The model seems to fit the data rather well. The deviance is decreasing and the dependence of the covariates approximates the true dependence as we see from the simulation model. What we want to prevent is that the model starts overfitting, We will perform a cross validation to choose the optimal number of trees. We observe that by sampling the folds at random becomes a problem as different folds can potentially have very high or low number of extremes. As the extremes act as levarage points in the process we have decided to use a stratified approach to the division in folds. Denote the ordered observations by $Y_{(1)} \geq \dots Y_{(n)}$, then for splitting in k folds each group of k observations is split among the folds, i.e. each variable in the sequence $(Y_{(k*j+1)}, Y_{(k*j+2)}, \dots, Y_{(k*j+k)}$ is assigned a different fold.

We can now perform a cross validation with the *CV_gbex* function. For this function we specify the maximum number of trees, which must be large enough such that we will not be underfitting the model.

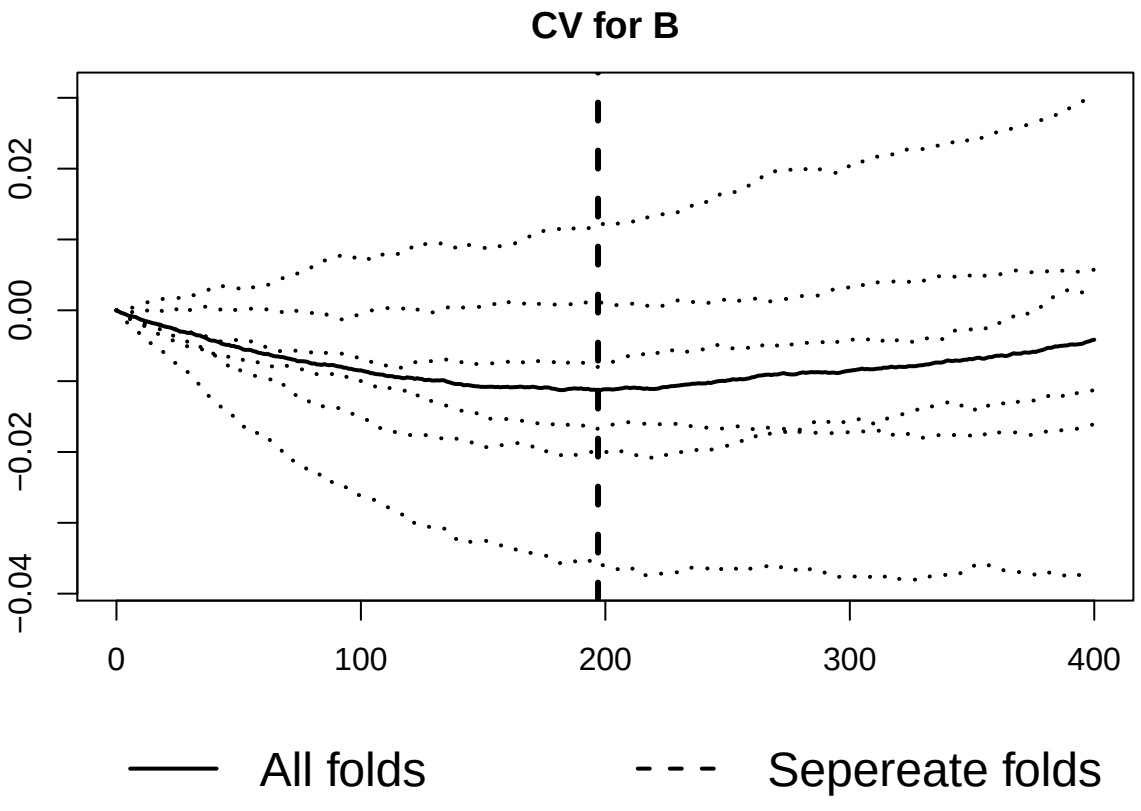
```
 CV_fit = CV_gbex(y,X,num_folds = 6,par="B", Bmax=400, stratified=T, lambda=lambda, depth=depth, min_lea
print(CV_fit)
```

```
## CV_gbex(y = y, X = X, num_folds = 6, par = "B", Bmax = 400, stratified = T,
```

```
##      lambda = lambda, depth = depth, min_leaf_size = min_leaf_size,
##      sf = sf, silent = T)
## A cross validation object for gbex for parameter B.
## The number of folds is 6 which are obtained by stratified sampling.
## The optimal parameter value is B = 197.
```

A more intricate look of the cross validation is obtained by plotting the deviance for the different folds,

```
plot(CV_fit,what="folds")
```



It is clear that the seperate folds are not clustered together, this can again be contributed to the extremes. A fold that does not behave well probably has a extreme observation Y for which there are no comparable observations in the training data with similar covariate values. As a result the deviance curves of each fold have each very different intercepts related to how well the extremes in this fold are represented by the models. To obtain an informative picture we set all intercepts to 0. Values lower (higher) than zero indicate an better (worse) performance compared to the unconditional model.

Many of the other parameters are hard to optimize, but the cross validation routine can also be used to try and optimize these parameters. One of the most important parameters is the *lambda* parameter. The σ and γ parameters depend in the optimization on each other. Taking too large steps with one will therefore negatively influence the other. For this reason we reperametrize λ by $\lambda_{ratio} = \frac{\lambda_{sigma}}{\lambda_{gamma}}$ and $\lambda_{scale} = \lambda_{sigma}$. By choosing λ_{scale} small enough we can use the cross validation function to find the optimal λ_{ratio}.

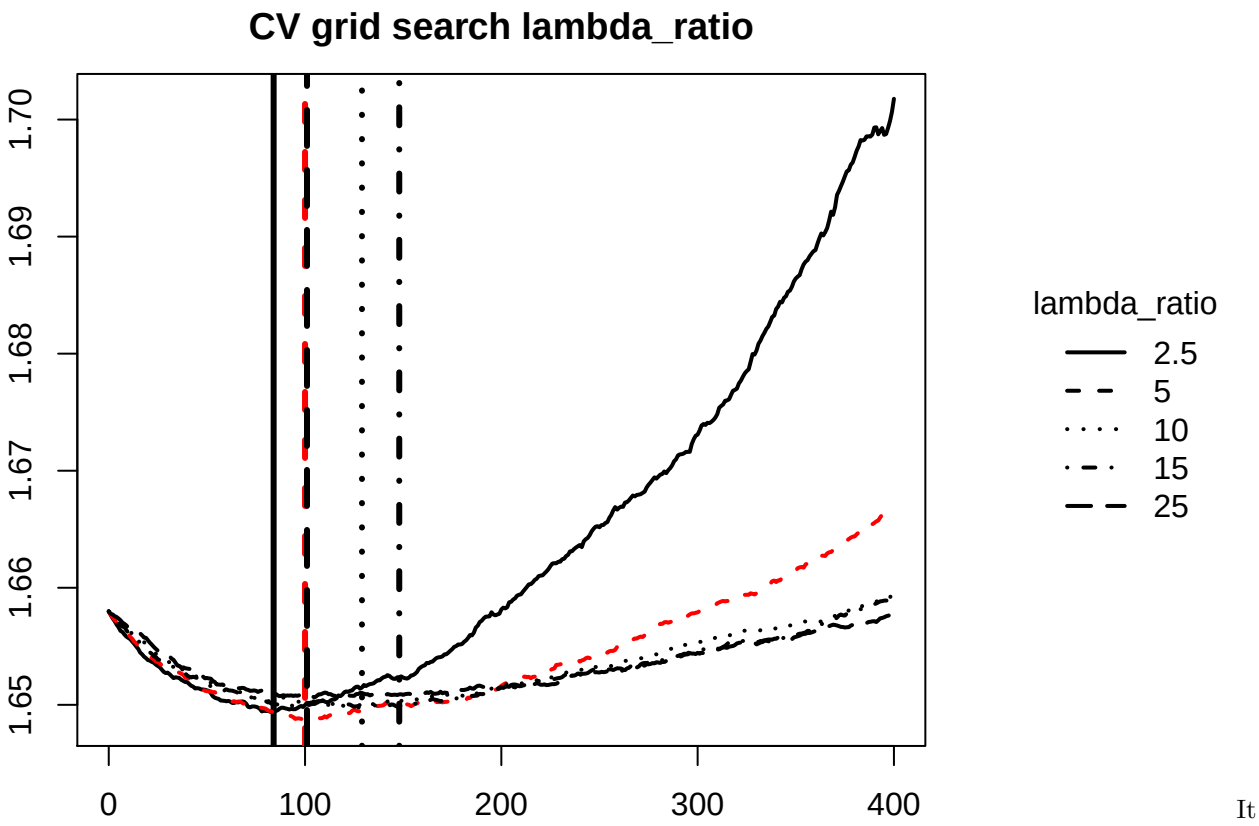
```
grid_lambda_ratio = c(2.5,5,10,15,25)
CV_lambda_ratio = CV_gbex(y,X,num_folds = 2,par="lambda_ratio", Bmax=400,grid = grid_lambda_ratio, stra
print(CV_lambda_ratio)
```

```
## CV_gbex(y = y, X = X, num_folds = 2, par = "lambda_ratio", Bmax = 400,
##      grid = grid_lambda_ratio, stratified = T, lambda_scale = 0.01,
##      depth = depth, min_leaf_size = min_leaf_size, sf = sf, silent = T)
## A cross validation object for gbex for parameter lambda_ratio.
```

```
## The number of folds is 2 which are obtained by stratified sampling.
## The optimal parameter value is lambda_ratio = 5.
```

Where we can again have a look at cross validation results for the different parameter values,

```
plot(CV_lambda_ratio,what="all")
```



It can be observed that different ratio's lead to different learning rates and therefore also the optimal number of trees is different. It is therefore important for each tuning parameter to allow for different values of B.

Some notes on the implementation of gbex

There are some choices made in the implementation that are worth discussing. First, it was observed that the algorithm is very unstable especially when tuning parameters are ill specified. As a result, it was observed that gradient steps sometimes are rather large leading to large decreases in deviance and sometimes negative σ parameter. Therefore we bound the absolute value of the gradient step by the learning rate for that parameter. Additionally, we reparamaterize $\sigma = \exp(\beta)$ and rewrote all derivatives in terms of the β parameter.