

# Package ‘GAMBoost’

January 2, 2012

**Version** 1.2-2

**Title** Generalized linear and additive models by likelihood based boosting

**Author** Harald Binder <binderh@fdm.uni-freiburg.de>

**Maintainer** Harald Binder <binderh@fdm.uni-freiburg.de>

**Depends** Matrix

**Suggests** snowfall, multicore

**Description** This package provides routines for fitting generalized linear and and generalized additive models by likelihood based boosting, using penalized B-splines

**License** GPL (>= 2)

**Repository** CRAN

**Date/Publication** 2011-09-20 08:29:11

## R topics documented:

cv.GAMBoost . . . . .	2
cv.GLMBoost . . . . .	4
estimPVal . . . . .	5
GAMBoost . . . . .	7
getGAMBoostSelected . . . . .	13
GLMBoost . . . . .	14
optimGAMBoostPenalty . . . . .	15
optimGLMBoostPenalty . . . . .	18
optimStepSizeFactor . . . . .	19
plot.GAMBoost . . . . .	21
predict.GAMBoost . . . . .	23
predict.GLMBoost . . . . .	24
<b>Index</b>	<b>26</b>

---

 cv.GAMBoost

*Cross-validation for GAMBoost fits*


---

### Description

Performs a K-fold cross-validation for [GAMBoost](#) in search for the optimal number of boosting steps.

### Usage

```
cv.GAMBoost(x=NULL,y,x.linear=NULL,subset=NULL,maxstepno=500,family=binomial(),
  weights=rep(1,length(y)),
  calc.hat=TRUE,calc.se=TRUE,trace=FALSE,parallel=FALSE,upload.x=TRUE,
  multicore=FALSE,folds=NULL,
  K=10,type=c("loglik","error","L2"),pred.cutoff=0.5,just.criterion=FALSE,...)
```

### Usage

```
cv.GAMBoost(x=NULL,y,x.linear=NULL,subset=NULL,maxstepno=500,
  K=10,type=c("loglik","error","L2"),pred.cutoff=0.5,
  just.criterion=FALSE,trace=FALSE,parallel=FALSE,
  upload.x=TRUE,multicore=FALSE,folds=NULL,...)
```

### Arguments

x	n * p matrix of covariates with potentially non-linear influence. If this is not given (and argument x.linear is employed), a generalized linear model is fitted.
y	response vector of length n.
x.linear	optional n * q matrix of covariates with linear influence.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
maxstepno	maximum number of boosting steps to evaluate.
K	number of folds to be used for cross-validation.
type, pred.cutoff	goodness-of-fit criterion: likelihood ("loglik"), error rate for binary response data ("error") or squared error for others ("L2"). For binary response data and the "error" criterion pred.cutoff specifies the p value cutoff for prediction of class 1 vs 0.
just.criterion	logical value indicating whether a list with the goodness-of-fit information should be returned or a GAMBoost fit with the optimal number of steps.
trace	logical value indicating whether information on progress should be printed.
parallel	logical value indicating whether computations in the cross-validation folds should be performed in parallel on a compute cluster, using package snowfall. Parallelization is performed via the package snowfall and the initialization function of of this package, sfInit, should be called before calling cv.GAMBoost.

multicore	indicates whether computations in the cross-validation folds should be performed in parallel, using package multicore. If TRUE, package multicore is employed using the default number of cores. A value larger than 1 is taken to be the number of cores that should be employed.
upload.x	logical value indicating whether x and x.linear should/have to be uploaded to the compute cluster for parallel computation. Uploading these only once (using sfExport(x,x.linear) from library snowfall) can save much time for large data sets.
folds	if not NULL, this has to be a list of length K, each element being a vector of indices of fold elements. Useful for employing the same folds for repeated runs.
...	miscellaneous parameters for the calls to <a href="#">GAMBoost</a>

### Value

GAMBoost fit with the optimal number of boosting steps or list with the following components:

criterion	vector with goodness-of fit criterion for boosting step 1, ..., maxstep
se	vector with standard error estimates for the goodness-of-fit criterion in each boosting step.
selected	index of the optimal boosting step.
folds	list of length K, where the elements are vectors of the indices of observations in the respective folds.

### Author(s)

Harald Binder <binderh@fdm.uni-freiburg.de>

### See Also

[GAMBoost](#)

### Examples

```
## Not run:
## Generate some data

x <- matrix(runif(100*8,min=-1,max=1),100,8)
eta <- -0.5 + 2*x[,1] + 2*x[,3]^2
y <- rbinom(100,1,binomial()$linkinv(eta))

## Fit the model with smooth components

gb1 <- GAMBoost(x,y,penalty=400,stepno=100,trace=TRUE,family=binomial())

## 10-fold cross-validation with prediction error as a criterion

gb1.crit <- cv.GAMBoost(x,y,penalty=400,maxstepno=100,trace=TRUE,
                      family=binomial(),
                      K=10,type="error",just.criterion=TRUE)
```

```
## Compare AIC and estimated prediction error

which.min(gb1$AIC)
which.min(gb1.crit$criterion)

## End(Not run)
```

---

cv.GLMBoost

*Cross-validation for GLMBoost fits*


---

### Description

Performs a convenience wrapper around [cv.GAMBoost](#) for performing a K-fold cross-validation for [GLMBoost](#) in search for the optimal number of boosting steps.

### Usage

```
cv.GLMBoost(x,y,penalty=length(y),just.criterion=TRUE,...)
```

### Arguments

y	response vector of length n.
x	n * q matrix of covariates with linear influence.
penalty	penalty for the covariates with linear influence.
just.criterion	logical value indicating whether a list with the goodness-of-fit information should be returned or a GLMBoost fit with the optimal number of steps.
...	parameters to be passed to <a href="#">cv.GAMBoost</a> or subsequently <a href="#">GAMBoost</a>

### Value

GLMBoost fit with the optimal number of boosting steps or list with the following components:

criterion	vector with goodness-of fit criterion for boosting step 1, ..., maxstep
se	vector with standard error estimates for the goodness-of-fit criterion in each boosting step.
selected	index of the optimal boosting step.

### Author(s)

Harald Binder <binderh@fdm.uni-freiburg.de>

### See Also

[GLMBoost](#), [cv.GAMBoost](#), [GAMBoost](#)

**Examples**

```
## Not run:
## Generate some data
x <- matrix(runif(100*8,min=-1,max=1),100,8)
eta <- -0.5 + 2*x[,1] + 4*x[,3]
y <- rbinom(100,1,binomial()$linkinv(eta))

## Fit the model with only linear components
gb1 <- GLMBoost(x,y,penalty=100,stepno=100,trace=TRUE,family=binomial())

## 10-fold cross-validation with prediction error as a criterion
gb1.crit <- cv.GLMBoost(x,y,penalty=100,maxstepno=100,trace=TRUE,
                        family=binomial(),
                        K=10,type="error")

## Compare AIC and estimated prediction error

which.min(gb1$AIC)
which.min(gb1.crit$criterion)

## End(Not run)
```

---

estimPVal

*Estimate p-values for a model fitted by GAMBoost or GLMBoost*


---

**Description**

Performs permutation-based p-value estimation for the optional covariates in a fit from [GAMBoost](#) or [GLMBoost](#). Currently binary response models with linear effects are supported, and the components have to be selected with `criterion="score"`

**Usage**

```
estimPVal(object,x,y,permute.n=10,per.covariate=FALSE,parallel=FALSE,
          multicore=FALSE,trace=FALSE,...)
```

**Arguments**

object	fit object obtained from <a href="#">GAMBoost</a> or <a href="#">GLMBoost</a> .
x	$n \times p$ matrix of covariates with linear effect. This has to be the same that was used as <code>x.linear</code> in the call to <a href="#">GAMBoost</a> or <code>x</code> in <a href="#">GLMBoost</a> .
y	response vector. This has to be the same that was used in the call to <a href="#">GAMBoost</a> or <a href="#">GLMBoost</a> .
permute.n	number of permutations employed for obtaining a null distribution.

<code>per.covariate</code>	logical value indicating whether a separate null distribution should be considered for each covariate. A larger number of permutations will be needed if this is wanted.
<code>parallel</code>	logical value indicating whether computations for obtaining a null distribution via permutation should be performed in parallel on a compute cluster. Parallelization is performed via the package <code>snowfall</code> and the initialization function of of this package, <code>sfInit</code> , should be called before calling <code>estimPVal</code> .
<code>multicore</code>	indicates whether computations in the permuted data sets should be performed in parallel, using package <code>multicore</code> . If <code>TRUE</code> , package <code>multicore</code> is employed using the default number of cores. A value larger than 1 is taken to be the number of cores that should be employed.
<code>trace</code>	logical value indicating whether progress in estimation should be indicated by printing the number of the permutation that is currently being evaluated.
<code>...</code>	miscellaneous parameters for the calls to <a href="#">GAMBoost</a>

### Details

As p-value estimates are based on permutations, random numbers are drawn for determining permutation indices. Therefore, the results depend on the state of the random number generator. This can be used to explore the variability due to random variation and help to determine an adequate value for `permute.n`. A value of 100 should be sufficient, but this can be quite slow. If there is a considerable number of covariates, e.g., larger than 100, a much smaller number of permutations, e.g., 10, might already work well. The estimates might also be negatively affected, if only a small number of boosting steps (say <50) was employed for the original fit.

### Value

Vector with p-value estimates, one value for each optional covariate with linear effect specified in the original call to [GAMBoost](#) or [GLMBoost](#).

### Author(s)

Harald Binder <binderh@fdm.uni-freiburg.de>

### References

Binder, H., Porzelius, C. and Schumacher, M. (2009). Rank-based p-values for sparse high-dimensional risk prediction models fitted by componentwise boosting. FDM-Preprint Nr. 101, University of Freiburg, Germany.

### See Also

[GAMBoost](#), [GLMBoost](#)

### Examples

```
## Not run:
## Generate some data
x <- matrix(runif(100*8,min=-1,max=1),100,8)
```

```

eta <- -0.5 + 2*x[,1] + 4*x[,3]
y <- rbinom(100,1,binomial()$linkinv(eta))

## Fit a model with only linear components
gb1 <- GLMBoost(x,y,penalty=100,stepno=100,trace=TRUE,family=binomial(),criterion="score")

# estimate p-values

p1 <- estimPVal(gb1,x,y,permute.n=10)

# get a second vector of estimates for checking how large
# random variation is

p2 <- estimPVal(gb1,x,y,permute.n=10)

plot(p1,p2,xlim=c(0,1),ylim=c(0,1),xlab="permute 1",ylab="permute 2")

## End(Not run)

```

---

GAMBoost

*Generalized additive model by likelihood based boosting*


---

## Description

GAMBoost is used to fit a generalized additive model by likelihood based boosting. It is especially suited for models with a large number of predictors with potentially non-linear influence. It provides smooth function estimates of covariate influence functions together with confidence bands and approximate degrees of freedom.

## Usage

```

GAMBoost(x=NULL,y,xmin=NULL,xmax=NULL,penalty=100,bdeg=2,pdiff=1,
  x.linear=NULL,standardize.linear=TRUE,
  penalty.linear=0,subset=NULL,
  criterion=c("deviance","score"),stepsize.factor.linear=1,
  sf.scheme=c("sigmoid","linear"),pendistmat.linear=NULL,
  connected.index.linear=NULL,
  weights=rep(1,length(y)),stepno=500,family=binomial(),
  sparse.boost=FALSE,sparse.weight=1,calc.hat=TRUE,calc.se=TRUE,
  AIC.type=c("corrected","classical"),return.score=TRUE,trace=FALSE)

```

## Arguments

x	n * p matrix of covariates with potentially non-linear influence. If this is not given (and argument x.linear is employed), a generalized linear model is fitted.
y	response vector of length n.

<code>xmin, xmax</code>	optional vectors of length $p$ specifying the lower and upper bound for the range of the smooth functions to be fitted.
<code>penalty</code>	penalty value for the update of an individual smooth function in each boosting step.
<code>bdeg, pdiff</code>	degree of the B-spline basis to be used for fitting smooth functions and difference of the coefficient estimates to which the penalty should be applied. When <code>pdiff</code> is a vector, the penalties corresponding to single vector elements are combined to enforce several types of smoothness simultaneously.
<code>x.linear</code>	optional $n * q$ matrix of covariates with linear influence.
<code>standardize.linear</code>	logical value indicating whether linear covariates should be standardized for estimation.
<code>penalty.linear</code>	penalty value (scalar or vector of length $q$ ) for update of individual linear components in each boosting step. If this is set to 0 the covariates in <code>x.linear</code> enter the model as mandatory covariates, which are updated together with the intercept term in each step.
<code>subset</code>	an optional vector specifying a subset of observations to be used in the fitting process.
<code>criterion</code>	Indicates the criterion to be employed for selecting the best update in each boosting step for the linear components. <code>deviance</code> corresponds to the original, deviance-based selection suggested for componentwise boosting. With <code>score</code> , a score statistic is employed. The latter is faster and corresponds to the approach employed for Cox regression in package <code>CoxBoost</code> .
<code>stepsize.factor.linear</code>	determines the step-size modification factor by which the natural step size of boosting steps for the linear covariates should be changed after a covariate has been selected in a boosting step. The default (value 1) implies constant penalties, for a value $< 1$ the penalty for a covariate is increased after it has been selected in a boosting step, and for a value $> 1$ the penalty it is decreased. If <code>pendistmat</code> is given, penalty updates are only performed for covariates that have at least one connection to another covariate.
<code>sf.scheme</code>	scheme for changing step sizes (via <code>stepsize.factor</code> ). "linear" corresponds to the scheme described in Binder and Schumacher (2009), "sigmoid" employs a sigmoid shape.
<code>pendistmat.linear</code>	connection matrix with entries ranging between 0 and 1, with entry $(i, j)$ indicating the certainty of the connection between covariates $i$ and $j$ . According to this information penalty changes due to <code>stepsize.factor.linear</code> $< 1$ are propagated, i.e., if entry $(i, j)$ is non-zero, the penalty for covariate $j$ is decreased after it has been increased for covariate $i$ , after it has been selected in a boosting step. This matrix either has to have dimension $q * q$ or the indices of the $q$ .connected connected linear covariates have to be given in <code>connected.index.linear</code> , in which case the matrix has to have dimension $q$ .connected $* q$ .connected.
<code>connected.index.linear</code>	indices of the $q$ .connected connected linear covariates, for which <code>pendistmat.linear</code> provides the connection information for distributing changes in penalties. If

	NULL, and a connection matrix is given, all covariates are assumed to be connected.
<code>weights</code>	an optional vector of weights to be used in the fitting process.
<code>stepno</code>	number of boosting steps ( $m$ ).
<code>family</code>	a description of the error distribution to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See <a href="#">family</a> for details of family functions.) Note that GAMBoost supports only canonical link functions and no scale parameter estimation, so <code>gaussian()</code> , <code>binomial()</code> and <code>poisson()</code> are the most plausible candidates.
<code>sparse.boost</code>	logical value indicating whether a criterion considering degrees of freedom (specifically AIC) should be used for selecting a covariate for an update in each boosting step (sparse boosting), instead of the deviance.
<code>sparse.weight</code>	factor modifying how the degrees of freedom enter into calculation of AIC for sparse boosting.
<code>calc.hat</code>	logical value indicating whether the hat matrix should be computed for each boosting step. If set to FALSE no degrees of freedom and therefore e.g. no AIC will be available. On the other hand fitting will be faster (especially for a large number of observations).
<code>calc.se</code>	logical value indicating whether confidence bands should be calculated. Switch this off for faster fitting.
<code>AIC.type</code>	type of model selection criterion to be calculated (and also to be used for sparse boosting if applicable): in the Gaussian case "classical" gives AIC and "corrected" results in corrected AIC (Hurvich, Simonoff and Tsai, 1998); for all other response families standard AIC is used.
<code>return.score</code>	logical value indicating whether the value of the score statistic, as evaluated in each boosting step for every covariate (only for binary response models, for linear components, if <code>criterion="score"</code> ), should be returned. The corresponding element <code>scoremat</code> can become very large (and needs much memory) when the number of covariates and boosting steps is large.
<code>trace</code>	logical value indicating whether progress in estimation should be indicated by printing the index of the covariate updated ( $1, \dots, p$ for smooth components and $p+1, \dots, p-q$ for parametric components) in the current boosting step.

## Details

The idea of likelihood based boosting (Tutz and Binder, 2006) is most easily understood for models with a Gaussian response. There it results in repeated fitting of residuals (This idea is then transferred to the generalized case). For obtaining an additive model GAMBoost uses a large number of boosting steps where in each step a penalized B-spline (of degree `bdeg`) (Eilers and Marx, 1996) is fitted to one covariate, the response being the residuals from the last step. The covariate to be updated is selected by deviance (or in case of sparse boosting by some model selection criterion). The B-spline coefficient estimates in each step are fitted under the constraint of a large penalty on their first (or higher order) differences. So in each step only a small adjustment is made. Summing over all steps for each covariate a smooth function estimate is obtained. When no basis expansion is used, i.e. just coefficients of covariates are updated, (generalized) linear models are obtained.

The main parameter of the algorithm is the number of boosting steps, given that the penalty is chosen large enough (If too small the minimum AIC will occur for a very early boosting step; see `optimGAMBoostPenalty`). When there is a large number of covariates with potentially non-linear effect, having a single parameter (with adaptive smoothness assignment to single components performed automatically by the algorithm) is a huge advantage compared to approaches, where a smoothing parameter has to be selected for each single component. The biggest advantage over conventional methods for fitting generalized additive models (e.g. `mgcv:gam` or `gam:gam`) will therefore be obtained for a large number of covariates compared to the number of observations (e.g. 10 covariates with only 100 observations). In addition GAMBoost performs well compared to other approaches when there is a small signal-to-noise ratio and/or a response (e.g. binary) with a small amount of information.

If a group of correlated covariates has influence on the response, e.g. genes from the same pathway, componentwise boosting will often result in a non-zero estimate for only one member of this group. To avoid this, information on the connection between covariates with linear influence can be provided in `pendistmat.linear`. If then, in addition, a penalty updating scheme with `stepsize.factor.linear < 1` is chosen, connected covariates with linear influence are more likely to be chosen in future boosting steps, if a directly connected covariate has been chosen in an earlier boosting step (see Binder and Schumacher, 2008b).

Note that the degrees of freedom (and based on these AIC and BIC) are just approximations, which are completely valid for example only when the order and the indices of the components updated is fixed. This leads to problems especially when there is a very large number of covariates (e.g. 10000 covariates with only 100 observations). Then it might be better (but also slower) to rely on cross validation (see `cv.GAMBoost`) for selection of the number of boosting steps.

Note that the `gamboost` routine in the R package `mboost` implements a different kind of boosting strategy: gradient based boosting instead of likelihood based boosting. The two approaches coincide only in special cases (e.g. L2 loss and Gaussian response). While gradient based boosting is more general, only likelihood based boosting allows e.g. for easily obtainable pointwise confidence bands.

## Value

GAMBoost returns an object of class GAMBoost. GAMBoost objects can be examined by `print`, `summary`, and `plot`. `getGAMBoostSelected` can be used on them to identify selected/significant covariates.

<code>x</code> , <code>n</code> , <code>p.linear</code>	original values for covariates with non-linear effect, number of observations, and number of covariates with linear effect.
<code>penalty</code> , <code>penalty.linear</code>	penalties used in updating smooth and linear components.
<code>stepno</code>	number of boosting steps.
<code>family</code>	response family.
<code>AIC.type</code>	type of AIC given in component AIC (applies only for Gaussian response).
<code>deviance</code> , <code>trace</code> , <code>AIC</code> , <code>BIC</code>	vectors of length <code>m</code> giving deviance, approximate degrees of freedom, AIC, and BIC for each boosting step.
<code>selected</code>	vector of length <code>m</code> given the index of the covariate updated in each boosting step (1-p for smooth components and (p+1), ... , (p-q) for parametric components).

beta	list of length $p+1$ , each element being a matrix with $m+1$ rows giving the estimated coefficients for the intercept term ( <code>beta[[1]]</code> ) and the smooth terms ( <code>beta[[2]]</code> , ... , <code>beta[[p+1]]</code> ).
beta.linear	$m * q$ matrix containing the coefficient estimates for the (standardized) linear covariates.
scoremat	$m * q$ matrix containing the value of the score statistic for each of the optional covariates before each boosting step, if selection has been performed by the score statistic ( <code>criterion="score"</code> ) and a binary response model has been employed.
mean.linear, sd.linear	vector of mean values and standard deviations used for standardizing the linear covariates.
hatmatrix	hat matrix at the final boosting step.
eta	$n * (m+1)$ matrix with predicted value (at predictor level) for each boosting step.
predictors	list of length $p+1$ containing information (as a list) on basis expansions for the smooth model components ( <code>predictors[[2]]</code> , ... , <code>predictors[[p+1]]</code> ).

**Author(s)**

Written by Harald Binder <binderh@fdm.uni-freiburg.de>, matching closely the original Fortran implementation employed for Tutz and Binder (2006).

**References**

- Binder, H. and Schumacher, M. (2009). Incorporating pathway information into boosting estimation of high-dimensional risk prediction models. *BMC Bioinformatics*. 10:18.
- Binder, H. and Schumacher, M. (2008). Incorporating pathway information into boosting estimation of high-dimensional risk prediction models. Manuscript.
- Hurvich, C. M., Simonoff, J. S. and Tsai, C. L. (1998). Smoothing parameter selection in nonparametric regression using and improved Akaike information criterion. *Journal of the Royal Statistical Society B*, **60**(2), 271–293.
- Eilers, P. H. C. and Marx, B. D. (1996) Flexible smoothing with B-splines and penalties. *Statistical Science*, **11**(2), 89–121.
- Tutz, G. and Binder, H. (2007) Boosting ridge regression. *Computational Statistics & Data Analysis*, **51**(12), 6044–6059.
- Tutz, G. and Binder, H. (2006) Generalized additive modelling with implicit variable selection by likelihood based boosting. *Biometrics*, **51**, 961–971.

**See Also**

[getGAMBoostSelected](#), [plot.GAMBoost](#), [predict.GAMBoost](#), [optimGAMBoostPenalty](#).

**Examples**

```

## Generate some data
n <- 100; p <- 8; q <- 2

# covariates with non-linear (smooth) effects
x <- matrix(runif(n*p,min=-1,max=1),n,p)

# binary covariates
x.linear <- matrix(round(runif(n*q,min=0,max=1)),n,q)

# 1st and 3rd smooth covariate and 1st linear covariate are informative
eta <- -0.5 + 2*x[,1] + 2*x[,3]^2 + x.linear[,1]-.5

y <- rbinom(n,1,binomial())$linkinv(eta))

## Fit a model with just smooth components
gb1 <- GAMBoost(x,y,penalty=500,stepno=100,family=binomial(),trace=TRUE)

# Inspect the AIC for a minimum
plot(gb1$AIC) # still falling at boosting step 100 so we need more steps
              # or a smaller penalty (use 'optimGAMBoostPenalty' for
              # automatic penalty optimization)

## Include two binary covariates as mandatory without penalty
## (appropriate for example for 'treatment/control')
## modelled as 'linear' predictors

gb2 <- GAMBoost(x,y,penalty=200,
                x.linear=x.linear,penalty.linear=0,
                stepno=100,family=binomial(),trace=TRUE)

## Include first binary covariates as mandatory and second
## as optional (e.g 'treatment/control' and 'female/male')

gb3 <- GAMBoost(x,y,penalty=200,
                x.linear=x.linear,penalty.linear=c(0,100),
                stepno=100,family=binomial(),trace=TRUE)

# Get summary with fitted covariates and estimates for
# the parametric components
summary(gb3)

# Extract boosted components at 'optimal' boosting step
selected <- getGAMBoostSelected(gb3,at.step=which.min(gb3$AIC))

# Plot all smooth components at final boosting step
par(mfrow=c(2,4))
plot(gb3)

# plot smooth components for which the null line is not inside the bands
# at 'optimal' boosting step (determined by AIC)
par(mfrow=c(1,length(selected$smoothbands)))

```

```

plot(gb3,select=selected$smoothbands,at.step=which.min(gb3$AIC))

## Fit a generalized linear model for comparison

x.linear <- cbind(x,x.linear)
gb4 <- GAMBoost(x=NULL,y=y,x.linear=x.linear,penalty.linear=100,
               stepno=100,trace=TRUE,family=binomial())

# Compare with generalized additive model fit
plot(0:100,gb3$AIC,type="l",xlab="stepno",ylab="AIC"); lines(0:100,gb4$AIC,lty=2)

## Fit a generalized linear model with penalty modification
## after every boosting step, with penalty changes being
## redistributed via a connection matrix

pendistmat <- matrix(0,10,10)
# Covariates 1 and 3 are connected
pendistmat[1,3] <- pendistmat[3,1] <- 1

gb5 <- GAMBoost(x=NULL,y=y,x.linear=x.linear,penalty.linear=100,
               stepsize.factor.linear=0.9,pendistmat.linear=pendistmat,
               stepno=100,trace=TRUE,family=binomial())

# or alternatively

gb5 <- GAMBoost(x=NULL,y=y,x.linear=x.linear,penalty.linear=100,
               stepsize.factor.linear=0.9,
               pendistmat.linear=pendistmat[c(1,3),c(1,3)],
               connected.index.linear=c(1,3),
               stepno=100,trace=TRUE,family=binomial())

```

---

getGAMBoostSelected *Identify selected/significant covariates from a GAMBoost object*

---

## Description

Extracts the information from a GAMBoost object which covariates have received any update up to a specific boosting step and for which smooth estimates the pointwise confidence bands do not contain the zero line.

## Usage

```
getGAMBoostSelected(object,at.step=NULL)
```

## Arguments

object	fitted GAMBoost object from a <a href="#">GAMBoost</a> call.
at.step	boosting step for which the information should be extracted. If none is given, the final boosting step is examined.

**Value**

smooth	indices of smooth components which received any update up to the given step.
smoothbands	indices of smooth components for which the pointwise confidence bands at the given step do not contain the null line (only available when confidence bands have been fitted; see option <code>se.fit</code> of <a href="#">GAMBoost</a> ).
parametric	indices of parametric components which received any update up to the given step.

**Author(s)**

Harald Binder <binderh@fdm.uni-freiburg.de>

**See Also**

[GAMBoost](#), [plot.GAMBoost](#)

**Examples**

```
## see examples for 'GAMBoost' and 'plot.GAMBoost'
```

---

GLMBoost

*Generalized linear model by likelihood based boosting*

---

**Description**

GLMBoost a convenience wrapper around [GAMBoost](#), for fitting generalized linear models by likelihood based boosting.

**Usage**

```
GLMBoost(x,y,penalty=length(y),standardize=TRUE,...)
```

**Arguments**

x	n * q matrix of covariates with linear influence.
y	response vector of length n.
penalty	penalty value (scalar or vector of length q) for update of individual linear components in each boosting step. If this is set to 0 the covariates enter the model as mandatory covariates, which are updated together with the intercept term in each step.
standardize	logical value indicating whether linear covariates should be standardized for estimation.
...	arguments that should be passed to <a href="#">GAMBoost</a>

**Value**

Object returned by call to `GAMBoost` (see documentation there), with additional class `GLMBoost`.

**Author(s)**

Harald Binder <binderh@fdm.uni-freiburg.de>

**References**

Tutz, G. and Binder, H. (2007) Boosting ridge regression. *Computational Statistics & Data Analysis*, **51**(12), 6044–6059.

**See Also**

`GAMBoost`, `predict.GLMBoost`.

**Examples**

```
## Generate some data
x <- matrix(runif(100*8,min=-1,max=1),100,8)
eta <- -0.5 + 2*x[,1] + 4*x[,3]
y <- rbinom(100,1,binomial()$linkinv(eta))

## Fit a model with only linear components
gb1 <- GLMBoost(x,y,penalty=100,stepno=100,trace=TRUE,family=binomial())

# Inspect the AIC for a minimum
plot(gb1$AIC)

# print the selected covariates, i.e., covariates with non-zero estimates
getGAMBoostSelected(gb1)

## Make the first two covariates mandatory

gb2 <- GLMBoost(x,y,penalty=c(0,0,rep(100,ncol(x)-2)),
               stepno=100,family=binomial(),trace=TRUE)
```

---

optimGAMBoostPenalty *Coarse line search for adequate GAMBoost penalty parameter*

---

**Description**

This routine helps in finding a penalty value that leads to an “optimal” number of boosting steps for `GAMBoost` (determined by AIC or cross-validation) that is not too small/in a specified range.

**Usage**

```
optimGAMBoostPenalty(x=NULL,y,x.linear=NULL,minstepno=50,maxstepno=200,start.penalty=500,method=c("
  penalty=100,penalty.linear=100,
  just.penalty=FALSE,iter.max=10,upper.margin=0.05,trace=TRUE,parallel=FALSE,
  calc.hat=TRUE,calc.se=TRUE,
  which.penalty=ifelse(!is.null(x),"smoothness","linear"),...)
```

**Usage**

```
optimGAMBoostPenalty(x=NULL,y,x.linear=NULL,
  minstepno=50,maxstepno=200,method=c("AICmin","CVmin"),
  which.penalty=ifelse(!is.null(x),"smoothness","linear"),
  start.penalty=500,penalty=100,penalty.linear=100,
  iter.max=10,upper.margin=0.05,
  just.penalty=FALSE,trace=TRUE,parallel=FALSE,...)
```

**Arguments**

<code>x</code>	<code>n * p</code> matrix of covariates with potentially non-linear influence. If this is not given (and argument <code>x.linear</code> is employed), a generalized linear model is fitted.
<code>y</code>	response vector of length <code>n</code> .
<code>x.linear</code>	optional <code>n * q</code> matrix of covariates with linear influence.
<code>minstepno</code> , <code>maxstepno</code>	range of boosting steps in which the “optimal” number of boosting steps is wanted to be.
<code>method</code>	determines how the optimal number of boosting steps corresponding to a fixed penalty is evaluated. With “AICmin” the AIC is used and with “CVmin” cross-validation is used as a criterion.
<code>which.penalty</code>	indicates whether the penalty for the smooth components (value “smoothness”) or for the linear components (“linear”) should be optimized.
<code>start.penalty</code>	start value for the search for the appropriate penalty.
<code>penalty</code> , <code>penalty.linear</code>	penalty values for the smooth or parametric components, respectively for the type of components for which the penalty is not optimized.
<code>just.penalty</code>	logical value indicating whether just the optimal penalty value should be returned or a <code>GAMBoost</code> fit performed with this penalty.
<code>iter.max</code>	maximum number of search iterations.
<code>upper.margin</code>	specifies the fraction of <code>maxstepno</code> which is used as an upper margin in which an AIC/cross-validation minimum is not taken to be one. (Necessary because of random fluctuations of these criteria).
<code>trace</code>	logical value indicating whether information on progress should be printed.
<code>parallel</code>	logical value indicating whether evaluation of cross-validation folds should be performed in parallel on a compute cluster, when using <code>method="CVmin"</code> . This requires library <code>snowfall</code> .

... miscellaneous parameters for [GAMBoost](#).

### Details

The penalty parameter(s) for [GAMBoost](#) have to be chosen only very coarsely. In Tutz and Binder (2006) it is suggested just to make sure, that the optimal number of boosting steps (according to AIC or cross-validation) is larger or equal to 50. With a smaller number of steps boosting may become too “greedy” and show sub-optimal performance. This procedure uses very a coarse line search and so one should specify a rather large range of boosting steps.

Penalty optimization based on AIC should work fine most of the time, but for a large number of covariates (e.g. 500 with 100 observations) problems arise and (more costly) cross-validation should be employed.

### Value

GAMBoost fit with the optimal penalty (with an additional component `optimGAMBoost.criterion` giving the values of the criterion (AIC or cross-validation) corresponding to the final penalty) or just the optimal penalty value itself.

### Author(s)

Written by Harald Binder <binderh@fdm.uni-freiburg.de>, matching closely the original Fortran implementation employed for Tutz and Binder (2006).

### References

Tutz, G. and Binder, H. (2006) Generalized additive modelling with implicit variable selection by likelihood based boosting. *Biometrics*, **51**, 961–971.

### See Also

[GAMBoost](#)

### Examples

```
## Not run:
## Generate some data

x <- matrix(runif(100*8,min=-1,max=1),100,8)
eta <- -0.5 + 2*x[,1] + 2*x[,3]^2
y <- rbinom(100,1,binomial()$linkinv(eta))

## Find a penalty (starting from a large value, here: 5000)
## that leads to an optimal number of boosting steps (based in AIC)
## in the range [50,200] and return a GAMBoost fit with
## this penalty

opt.gb1 <- optimGAMBoostPenalty(x,y,minstepno=50,maxstepno=200,
                               start.penalty=5000,family=binomial(),
                               trace=TRUE)
```

```
# extract the penalty found/used for the fit
opt.gb1$penalty

## End(Not run)
```

---

optimGLMBoostPenalty *Coarse line search for adequate GLMBoost penalty parameter*

---

### Description

This routine is a convenience wrapper around [optimGAMBoostPenalty](#) for finding a penalty value that leads to an “optimal” number of boosting steps for GLMBoost (determined by AIC or cross-validation) that is not too small/in a specified range.

### Usage

```
optimGLMBoostPenalty(x,y,start.penalty=length(y),just.penalty=FALSE,...)
```

### Arguments

x	n * q matrix of covariates with linear influence.
y	response vector of length n.
start.penalty	start value for the search for the appropriate penalty.
just.penalty	logical value indicating whether just the optimal penalty value should be returned or a <a href="#">GLMBoost</a> fit performed with this penalty.
...	miscellaneous parameters for <a href="#">optimGAMBoostPenalty</a> .

### Details

The penalty parameter(s) for [GLMBoost](#) have to be chosen only very coarsely. In Tutz and Binder (2006) it is suggested just to make sure, that the optimal number of boosting steps (according to AIC or cross-validation) is larger or equal to 50. With a smaller number of steps boosting may become too “greedy” and show sub-optimal performance. This procedure uses very a coarse line search and so one should specify a rather large range of boosting steps.

Penalty optimization based on AIC should work fine most of the time, but for a large number of covariates (e.g. 500 with 100 observations) problems arise and (more costly) cross-validation should be employed.

### Value

GLMBoost fit with the optimal penalty (with an additional component `optimGAMBoost.criterion` giving the values of the criterion (AIC or cross-validation) corresponding to the final penalty) or just the optimal penalty value itself.

**Author(s)**

Written by Harald Binder <binderh@fdm.uni-freiburg.de>, matching closely the original Fortran implementation employed for Tutz and Binder (2006).

**References**

Tutz, G. and Binder, H. (2006) Generalized additive modelling with implicit variable selection by likelihood based boosting. *Biometrics*, **51**, 961–971.

**See Also**

[GLMBoost](#), [optimGLMBoostPenalty](#), [GAMBoost](#)

**Examples**

```
## Not run:
## Generate some data

## Generate some data
x <- matrix(runif(100*8,min=-1,max=1),100,8)
eta <- -0.5 + 2*x[,1] + 4*x[,3]
y <- rbinom(100,1,binomial()$linkinv(eta))

## Find a penalty (starting from a large value, here: 5000)
## that leads to an optimal number of boosting steps (based in AIC)
## in the range [50,200] and return a GLMBoost fit with
## this penalty

opt.gb1 <- optimGLMBoostPenalty(x,y,minstepno=50,maxstepno=200,
                               start.penalty=5000,family=binomial(),
                               trace=TRUE)

# extract the penalty found/used for the fit
opt.gb1$penalty

## End(Not run)
```

---

optimStepSizeFactor    *Coarse line search for optimum step-size modification factor*

---

**Description**

This routine helps in finding an optimum step-size modification factor for [GAMBoost](#), i.e., that results in an optimum in terms of cross-validated log-likelihood.

**Usage**

```
optimStepSizeFactor(x=NULL,y,x.linear=NULL,
                    direction=c("down","up","both"),start.stepsize=0.1,
                    iter.max=10,constant.cv.res=NULL,parallel=FALSE,
                    trace=FALSE,...)
```

**Arguments**

<code>x</code>	<code>n * p</code> matrix of covariates with potentially non-linear influence. If this is not given (and argument <code>x.linear</code> is employed), a generalized linear model is fitted.
<code>y</code>	response vector of length <code>n</code> .
<code>x.linear</code>	optional <code>n * q</code> matrix of covariates with linear influence.
<code>direction</code>	direction of line search for an optimal step-size modification factor (starting from value 1).
<code>start.stepsize</code>	step size used for the line search. A final step is performed using half this size.
<code>iter.max</code>	maximum number of search iterations.
<code>constant.cv.res</code>	result of <code>cv.GAMBoost</code> (with <code>just.criterion=TRUE</code> ) for <code>stepsize.factor.linear=1</code> , that can be provided for saving computing time, if it already is available.
<code>parallel</code>	logical value indicating whether evaluation of cross-validation folds should be performed in parallel on a compute cluster. This requires library <code>snowfall</code> .
<code>trace</code>	logical value indicating whether information on progress should be printed.
<code>...</code>	miscellaneous parameters for <code>cv.GAMBoost</code> .

**Details**

A coarse line search is performed for finding the best parameter `stepsize.factor.linear` for `GAMBoost`. If an `pendistmat.linear` argument is provided (which is passed on to `GAMBoost`), a search for factors smaller than 1 is sensible (corresponding to `direction="down"`). If no connection information is provided, it is reasonable to employ `direction="both"`, for avoiding restrictions without subject matter knowledge.

**Value**

List with the following components:

<code>factor.list</code>	array with the evaluated step-size modification factors.
<code>critmat</code>	matrix with the mean log-likelihood for each step-size modification factor in the course of the boosting steps.
<code>optimal.factor.index</code>	index of the optimal step-size modification factor.
<code>optimal.factor</code>	optimal step-size modification factor.
<code>optimal.step</code>	optimal boosting step number, i.e., with minimum mean log-likelihood, for step-size modification factor <code>optimal.factor</code> .

**Author(s)**

Written by Harald Binder <binderh@fdm.uni-freiburg.de>.

**References**

Binder, H. and Schumacher, M. (2009). Incorporating pathway information into boosting estimation of high-dimensional risk prediction models. *BMC Bioinformatics*. 10:18.

**See Also**

[GAMBoost](#), [cv.GAMBoost](#)

**Examples**

```
## Not run:
## Generate some data
n <- 100; p <- 10

# covariates with non-linear (smooth) effects
x <- matrix(runif(n*p,min=-1,max=1),n,p)
eta <- -0.5 + 2*x[,1] + 2*x[,3]^2 + x[,9]-.5
y <- rbinom(n,1,binomial()$linkinv(eta))

# Determine step-size modification factor for a generalize linear model
# As there is no connection matrix, perform search into both directions

optim.res <- optimStepSizeFactor(direction="both",
                                y=y,x.linear=x,family=binomial(),
                                penalty.linear=200,
                                trace=TRUE)

# Fit with obtained step-size modification parameter and optimal number of boosting
# steps obtained by cross-validation

gb1 <- GAMBoost(x=NULL,y=y,x.linear=x,family=binomial(),penalty.linear=200,
               stepno=optim.res$optimal.step,
               stepsize.factor.linear=optim.res$optimal.factor)

summary(gb1)

## End(Not run)
```

---

plot.GAMBoost

*Plots of the smooth functions from a GAMBoost fit*

---

**Description**

Generates plots for the smooth components from a [GAMBoost](#) fit at a specific boosting step.

**Usage**

```
\bsl{method}\{plot\}\{GAMBoost\}(x,select=NULL,at.step=NULL,add=FALSE,phi=1,ylim=NULL,xlab=NULL,ylabel=NULL)
```

**Usage**

```
## S3 method for class 'GAMBoost'
plot(x,select=NULL,at.step=NULL,add=FALSE,phi=1,...)
```

**Arguments**

x	fitted GAMBoost object from a <a href="#">GAMBoost</a> call.
select	indices of the smooth component(s) for which plots should be generated. If none are specified, all are used.
at.step	boosting step from which the estimates for the smooth functions should be evaluated. If not given, the final boosting step is used.
add	logical value indicating whether the plot(s) should be added to the current plot.
phi	scale parameter for the confidence bands.
...	miscellaneous plotting parameters given to the low level plotting routine.

**Value**

A plot is produced for the specified smooth components in the GAMBoost fit. Pointwise confidence bands are plotted when the standard error information has been calculated (option `calc.se=TRUE` in the call to GAMBoost).

**Author(s)**

Harald Binder <binderh@fdm.uni-freiburg.de>

**Examples**

```
## Generate some data

x <- matrix(runif(100*8,min=-1,max=1),100,8)
eta <- -0.5 + 2*x[,1] + 2*x[,3]^2
y <- rbinom(100,1,binomial()$linkinv(eta))

## Fit the model with smooth components
gb1 <- GAMBoost(x,y,penalty=400,stepno=100,trace=TRUE,family=binomial())

## Plot smooth components of fit

# all, at final boosting step
par(mfrow=c(2,4))
plot(gb1)

# components that received an update up to the 'optimal' boosting step
```

```

selected <- getGAMBoostSelected(gb1,at.step=which.min(gb1$AIC))

par(mfrow=c(1,length(selected$smooth)))
plot(gb1,select=selected$smooth)

# components where the estimate at the 'optimal' boosting step does not
# contain the null line
par(mfrow=c(1,length(selected$smoothbands)))
plot(gb1,select=selected$smoothbands)

```

---

predict.GAMBoost      *Predict method for GAMBoost fits*

---

### Description

Obtains predictions at specified boosting steps from a GAMBoost object fitted by [GAMBoost](#).

### Usage

```

## S3 method for class 'GAMBoost'
predict(object,newdata=NULL,newdata.linear=NULL,
        at.step=NULL,type=c("link","response","terms"),...)

```

### Arguments

object	fitted GAMBoost object from a <a href="#">GAMBoost</a> call.
newdata	n.new * p matrix with new covariate values for smooth components. If just prediction for the training data is wanted or just a generalized linear model has been fitted, it can be omitted.
newdata.linear	matrix with new covariate values for linear components. If linear components have been fitted and this is not given, the contribution of the linear components will be ignored for prediction.
at.step	scalar or vector of boosting step(s) at which prediction is wanted. If type="terms" is used, only one step is admissible. If no step is given, the final boosting step is used.
type	type of prediction to be returned: "link" gives prediction at the level of the predictor, "response" at the response level. "terms" returns individual contributions of the smooth components to the predictor.
...	miscellaneous arguments, none of which is used at the moment.

### Value

For type="link" and type="response" a vector of length n.new (at.step being a scalar) or a n.new \* length(at.step) matrix (at.step being a vector) with predictions is returned. For type="terms" a n.new \* p+1 matrix with contributions of the smooth components to the predictor is returned.

**Author(s)**

Harald Binder <binderh@fdm.uni-freiburg.de>

**Examples**

```
## Generate some data
x <- matrix(runif(100*3,min=-1,max=1),100,3)
eta <- -0.5 + 2*x[,1] + 4*x[,3]^2
y <- rbinom(100,1,binomial()$linkinv(eta))

## Fit the model with smooth components
gb1 <- GAMBoost(x,y,penalty=200,stepno=100,trace=TRUE,family=binomial())

## Extract predictions

# at final boosting step
predict(gb1,type="response")

# at 'optimal' boosting step (with respect to AIC)
predict(gb1,at.step=which.min(gb1$AIC),type="response")

# matrix with predictions at predictor level for all boosting steps
predict(gb1,at.step=1:100,type="link")
```

---

predict.GLMBoost      *Predict method for GLMBoost fits*

---

**Description**

Convenience wrapper for [predict.GAMBoost](#), for obtaining predictions at specified boosting steps from a GAMBoost object fitted by [GLMBoost](#).

**Usage**

```
## S3 method for class 'GLMBoost'
predict(object,newdata=NULL,...)
```

**Arguments**

object	fitted GAMBoost object from a <a href="#">GLMBoost</a> call.
newdata	n.new * q matrix with new covariate values for linear components. If just prediction for the training data is wanted, it can be omitted.
...	arguments that should be passed to <a href="#">predict.GAMBoost</a> .

**Value**

Value returned by [predict.GAMBoost](#) (see documentation there).

**Author(s)**

Harald Binder <binderh@fdm.uni-freiburg.de>

**See Also**

[GLMBoost](#), [GAMBoost](#), [predict.GAMBoost](#).

**Examples**

```
## Generate some data
x <- matrix(runif(100*8,min=-1,max=1),100,8)
eta <- -0.5 + 2*x[,1] + 4*x[,3]
y <- rbinom(100,1,binomial()$linkinv(eta))

## Fit the model with only linear components
gb1 <- GLMBoost(x,y,penalty=100,stepno=100,trace=TRUE,family=binomial())

## Extract predictions

# at final boosting step
predict(gb1,type="response")

# at 'optimal' boosting step (with respect to AIC)
predict(gb1,at.step=which.min(gb1$AIC),type="response")

# matrix with predictions at predictor level for all boosting steps
predict(gb1,at.step=1:100,type="link")
```

# Index

## \*Topic **models**

- cv.GAMBoost, 2
- cv.GLMBoost, 4
- estimPVal, 5
- GAMBoost, 7
- getGAMBoostSelected, 13
- GLMBoost, 14
- optimGAMBoostPenalty, 15
- optimGLMBoostPenalty, 18
- optimStepSizeFactor, 19
- plot.GAMBoost, 21
- predict.GAMBoost, 23
- predict.GLMBoost, 24

## \*Topic **regression**

- cv.GAMBoost, 2
- cv.GLMBoost, 4
- estimPVal, 5
- GAMBoost, 7
- getGAMBoostSelected, 13
- GLMBoost, 14
- optimGAMBoostPenalty, 15
- optimGLMBoostPenalty, 18
- optimStepSizeFactor, 19
- plot.GAMBoost, 21
- predict.GAMBoost, 23
- predict.GLMBoost, 24

## \*Topic **smooth**

- cv.GAMBoost, 2
- GAMBoost, 7
- getGAMBoostSelected, 13
- optimGAMBoostPenalty, 15
- optimStepSizeFactor, 19
- plot.GAMBoost, 21
- predict.GAMBoost, 23

## \*Topic **survial**

- estimPVal, 5

estimPVal, 5

family, 9

GAMBoost, 2–6, 7, 13–17, 19–23, 25  
getGAMBoostSelected, 10, 11, 13  
GLMBoost, 4–6, 14, 18, 19, 24, 25

optimGAMBoostPenalty, 10, 11, 15, 18, 19  
optimGLMBoostPenalty, 18  
optimStepSizeFactor, 19

plot.GAMBoost, 11, 14, 21  
predict.GAMBoost, 11, 23, 24, 25  
predict.GLMBoost, 15, 24

cv.GAMBoost, 2, 4, 10, 20, 21  
cv.GLMBoost, 4