

Package ‘gbev’

February 9, 2012

Version 0.1.1

Date 2009-02-24

Title Gradient Boosted Regression Trees with Errors-in-Variables

Author Joe Sexton <j.a.sexton@medisin.uio.no>

Maintainer Joe Sexton <j.a.sexton@medisin.uio.no>

Depends mvtnorm

Description This package performs non-parametric regression when covariates are measured with error. The models are estimated using gradient boosted regression trees. Regression is performed using squared error loss, while binary response regression can be performed using negative log-likelihood loss.

License GPL (>= 2)

Repository CRAN

Date/Publication 2009-02-24 08:55:24

R topics documented:

checkMeasErrorModel	2
cvLoss	2
gbev	3
gbev.fit	8
mixModelMatrices	9
part.dep	10
plotLoss	11
predict_gbev	12
treeMatrix	12
var.imp	13

Index	14
--------------	-----------

checkMeasErrorModel *Helper function for gbev-function.*

Description

A help function for checking the specification of the measurement error model.

Usage

```
checkMeasErrorModel(measErrorModel)
```

Arguments

measErrorModel A list of components specifying the measurement error model.

cvLoss *Cross-validation of boosting iteration.*

Description

Performs k-fold cross-validation of number of boosting iterations.

Usage

```
cvLoss(object,k,random=F,loss="logLike")
```

Arguments

object	A fitted model of type gbev.object.
k	Is the k in k-fold cross-validation.
random	If TRUE then cross-validation is done by randomly sampling (without replacement) the validation group, else the validation groups are determined from order appearing in data.
loss	Can be logLike for binary regression (negative log-likelihood loss), or L2 for squared error loss.

Value

Returns a list containing, `iters` which are iterations the cross-validation loss is evaluated, `cvLoss` which is the cross-validated loss-function, `estIter` which is the iteration minimizing the cross-validated loss.

gbev

*Boosted regression trees with errors-in-variables***Description**

Fits boosted regression trees with errors-in-variables.

Usage

```
gbev(formula = formula(data),
     data = list(),
     weights=NULL,
     measErrorModel=NULL,
     method="L2",
     indepFitUpdate=1,
     nboost=100,
     lambda=100,
     maxDepth=2,
     minSplit=10,
     minBucket=0,
     sPoints=10,
     mc=2,
     intermPred=10,
     maxSplitAttempts=10)
```

Arguments

formula	A symbolic description of the model to be fit. NOTE: the error measured covariates must be placed first in the formula.
data	A data frame containing variables in the model.
weights	Weights applied to observations. Defaults to 1.
measErrorModel	This is a list specifying the distribution of the latent covariates and the measurement error. Here it is assumed that the latent covariates are a mixture of normals (possibly multivariate), and that the measurement error is normally distributed. See examples below for details. Note that at least one covariate must be specified as measured with error.
method	Can be L2 for squared error loss, logLike for binary regression with negative log-likelihood loss.
indepFitUpdate	If indepFitUpdate=1 then model fit is updated using independent MC-sample, else the fit is updated using the same MC-sample as used in tree fitting.
nboost	Number of boosting iterations to perform.
lambda	Regularization parameter.
maxDepth	Determines maximum interaction depth of trees fit. maxDepth=2 fits stumps.

<code>minSplit</code>	Minimum expected number of observations in a node for it to be split.
<code>minBucket</code>	Minimum expected number of observations in a node.
<code>sPoints</code>	Number of points sampled at random from which to choose split.
<code>mc</code>	Number of Monte-Carlo samples drawn to compute node probabilities.
<code>intermPred</code>	Increments of iterations at which intermediate predictions are saved, required for <code>cvLoss</code> function.
<code>maxSplitAttempts</code>	Maximum number of attempts at finding a valid split point. When splitting a node, <code>sPoints</code> candidate splits are supposed to be found for each covariate, however, each randomly sampled split point does not necessarily give a valid split point (i.e. a point satisfying <code>minBucket</code> and <code>minSplit</code>), and <code>maxSplitAttempts</code> is the maximum number of attempts at finding such a point. .

Details

This function performs non-parametric regression when some or all covariates are measured with error. It is assumed that the latent (error measured) covariate vector, X , is observed via a random variable W , the relation between the two being

$$W = X + U$$

where U is the measurement error. This function assumes that the density of X is known and representable as a finite mixture of multivariate normal densities, while the measurement error U is assumed to be multivariate normal. These two densities are specified in the `measErrorModel` argument of the `gbev` function, see examples below for further details on `measErrorModel`. In practice the densities of X and U will not be known, but must be estimated in some manner. This is typically done using replicate measurements or validation data, see Carroll et.al. (1995) for further details.

It is assumed that in the `formula` argument the error measured covariates occur first. That is, for example, if one has covariates $W1, W2, Z1$ and $Z2$, where the first two are error contaminated and the last two error-free then `formula` must be `y~w1+w2+z1+z2`.

The function `gbev` estimates $E(Y|X)$ where Y is the response, using boosted regression trees. That is, a model $F(x)$ for $E(Y|x)$ is estimated such that $F(x) = \sum_{k=1}^K T_k(x)$ where each $T_k(x)$ is a regression tree, and each regression tree is estimated by fitting it to the gradient of the previous model estimate using least squares, as described in Friedman (2001).

The model $F(x)$ is built up iteratively, such that at each boosting iteration a tree model is added to $F(x)$ to minimize the squared error between the observed data model, given by $E(F(X)|w)$, and the current model residuals. More specifically, on the k -th iteration, the observed data residuals are $\tilde{y}_i = (y_i - E(F_k(X)|w))$ for $i = 1, \dots, n$, and a tree $T_k(x) = \sum_{j=1}^J c_{kj} I(x \in R_{kj})$ is fit by fitting $E(T_k(X)|w)$, using least squares and recursive partitioning, to the current residuals \tilde{y}_i . The model estimate is then updated by setting $F_{k+1}(x) = F_k(x) + T_k(x)$, and the procedure iterated.

The tree fitting requires evaluating probabilities of the form $Pr(X \in R|w) = E(I(X \in R)|w)$, i.e. the probability that the covariate X is in a rectangular region R , given that $W = w$. These probabilities are here estimated using Monte Carlo sampling, and the argument `mc` in `gbev` regulates how many samples are drawn, at each iteration, for each observation, the default being `mc=2`. Boosting is known to perform best when the amount of fitting at each iteration is small, here this is regulated by the size of the regression trees, the `maxDepth` argument, and the regularization parameter `lambda`.

The larger values of `lambda` the less fitting is done per boosting iteration, and the more iterations are typically required to achieve adequate fitting. The `lambda` parameter is also used to control the Monte Carlo error in the function estimate, due to the sampling in the tree fitting. It turns out that the larger value of `lambda` the smaller Monte Carlo error is in the regression function estimate, for the same value of `mc`. Some experimentation is required to find an appropriate `lambda` value such that the Monte Carlo error is acceptable.

Two loss functions are available for estimation, specified in the `method` argument. Setting `method="L2"` squared error loss is used, while `method="logLike"` negative log-likelihood loss is used for performing binary, Y is 0 or 1, regression.

The arguments `maxDepth`, `minSplit`, `minBucket`, and `sPoints` control the tree fitting. `maxDepth` controls the depth of the regression tree to be fit, with `maxDepth=2` fitting a tree containing a single split (two terminal nodes), `maxDepth=3` fits a tree with 4 terminal nodes, obtained by splitting the nodes of a single split tree. Interestingly, `maxDepth=2` fits an additive model, `maxDepth=3` a model with all second order interactions, `maxDepth=4` a model with all third order interactions, and so on. The argument `sPoints` is the number of candidate split points sampled for each covariate when splitting a node. In the absence of measurement error node splitting is typically done by examining all values of the observed covariates falling in the node being split, however with measurement error the realized values of X are not observed, and sampling is here used to generate candidate split points, with `sPoints` governing the number of such split points sampled for each covariate. Interestingly, smaller values of `sPoints` often work the best. The argument `minSplit` determines the smallest (expected) number of observations in a node for a split to be attempted, while the argument `minBucket` the smallest (expected) number of observations in a terminal node. Note that using non-zero values of `lambda`, `minBucket=0` will run without error.

Value

`gbev` returns `gbev` object.

Author(s)

Joe Sexton <j.a.sexton@medisin.uio.no>

References

J.H. Friedman (2001). "Greedy Function Approximation: A Gradient Boosting Machine," *Annals of Statistics* 29(5):1189-1232.

T. Hastie, R. Tibshirani and J.H. Friedman (2001). "The Elements of Statistical Learning" Springer.

R. Carroll, D. Ruppert and L. Stenfanski (1995). "Measurement Error in Nonlinear Models," Chapman and Hall.

Examples

```
### Univariate regression example
n<-500
varX<-1
varME<-0.25
varNoise<-0.3^2
```

```

### Data
x<-rnorm(n,sd=sqrt(varX))          ### Error free covariate
w<-x+rnorm(n,sd=sqrt(varME))      ### Error contaminated version
fx<-sin(pi*x/2)/(1+2*(x^2)*((2*as.numeric(x>=0)-1)+1))  ### True regression function
y<-fx+rnorm(n,sd=sqrt(varNoise))  ### Response
dat<-data.frame(y=y,w=w)

### Measurement error model ####
###
### The measurement error model is a list of the following components:
###
### SigmaX:   the covariance matrices of the mixture model for the error free covariates
###           SigmaX[i,,] is the covariance matrix of the i-th mixture density
### mu:       the means of the mixture model for the error free covariates
###           mu[i,] is the mean-vector of the i-th mixture density
### SigmaME:  the covariance matrix of the measurment error
### pComp:    the weights of the mixture distribution, pComp[i] is the weight of the
###           i-th mixture density
### numComp:  the number of components in the mixture
##
p<-1
pME<-1

numComp<-3          ## number of components in gaussian mixture for X-distribution
SigmaME<-diag(varME,pME)
SigmaJ<-array(dim=c(numComp,pME,pME))
mu<-array(dim=c(numComp,pME))
pComp<-array(1/numComp,dim=c(numComp,1))
for(i in 1:numComp)
{
SigmaJ[i,,]<-diag(varX,pME)
mu[i,]<-rep(0,pME)
}
### list required by "gbev" for measurement error model
meModel<-list(SigmaX=SigmaJ,mu=mu,SigmaME=SigmaME,pComp=pComp,numComp=numComp)

fit<-gbev(y~w,data=dat,
          measErrorModel=meModel,
          method="L2",          ## Squared error loss
          nboost=1000,         ## 1000 boosting iterations
          lambda=5,           ## regularization of regression tree
          maxDepth=2,         ## maximum tree depth, 2 corresponds stumps
          mc=2,                ## number of monte-carlo samples per tree build
          minSplit=3,         ## minimum number of obs in node to split
          minBucket=0,        ## minimum number of obs in nodes
          sPoints=10,         ## number of sampled candidate split points
          intermPred=5)       ## increments of iterations to store predictions

### 5-fold cross-validation
hcv<-cvLoss(object=fit,k=5,random=FALSE,loss="L2")
plot(hcv$iters,hcv$cvLoss,type="l")

```

```

hp<-part.dep(object=fit,varIndx=1,firstTree=1,lastTree=hcv$estIter)

x<-seq(-2,2,by=.02)
fx<-sin(pi*x/2)/(1+2*(x^2)*((2*as.numeric(x>=0)-1)+1))
points(x,fx,type="l",lty=5)

## Simulated binary regression example,
## with: Y=I( X1*X2+X2*X3+X1*X3>0), with measurement error on X's
n<-1000
p<-3
varX<-1 ##
varME<-0.5 ## measurement error variance

x<-rnorm(p*n)
x<-matrix(x,ncol=p,nrow=n)
## add measurement error
w<-x+matrix(rnorm(p*n,sd=sqrt(varME)),ncol=p,nrow=n)

x<-x[,c(1:p)]*x[,c(2:p,1)]
x<-apply(x,1,sum)
threshold<-0
y<-as.numeric(x>threshold)
dat<-data.frame(y=y,w1=w[,1],w2=w[,2],w3=w[,3]) ## must be modified if(p!=3)

#### Measurement error model #####
numComp<-1 ## Number of components in mixture
SigmaME<-diag(varME,p) ## Covariance matrix of measurement error
SigmaJ<-array(dim=c(numComp,p,p)) ## Covariance matrices for mixture
mu<-array(dim=c(numComp,p)) ## Mean vectors for mixture components
pComp<-array(1/numComp,dim=c(numComp,1)) ## Mixture probabilities
for(i in 1:numComp)
{ ## filling in mixture model for X-distribution
SigmaJ[i,,]<-diag(varX,p)
mu[i,]<-rep(0,p)
}
## The list for measurement error model
meModel<-list(SigmaX=SigmaJ,mu=mu,SigmaME=SigmaME,pComp=pComp,numComp=numComp)

fit<-gbev(y~w1+w2+w3,data=dat,
measErrorModel=meModel,
method="logLike", ## loss function
nboost=1000, ## number of boosting iterations
lambda=40, ## regularization parameter used in regression tree
maxDepth=3, ## maximum depth of regression tree
minSplit=10, ## minimum number of observations in node to split
minBucket=0, ## minimum number in split node to allow split
sPoints=2, ## number of sampled candidate split points
mc=2, ## monte-carlo sample size used in each regression tree
intermPred=10) ## Increments of iterations to store loss function

```

```
## plot loss function as function of iterations
hp<-plotLoss(fit,loss="logLike",startIter=10)

## bivariate partial dependence plot
hdp<-part.dep(object=fit,varIndx=c(1,2),firstTree=1,
lastTree=1000,ngrid=50)
dpp<-data.frame(x1=hdp$dat$x,x2=hdp$dat$y,prob=hdp$dat$z)
library(lattice)
wireframe(prob~x1*x2,dpp,aspect=c(1,0.5),drape=TRUE,screen=list(z=50,x=-60),
scales=list(arrows=FALSE),xlim=c(-2.5,2.5),ylim=c(-2.5,2.5))
```

gbev.fit

Helper function for gbev-function.

Description

Help function for fitting boosted regression trees with errors-in-variables.

Usage

```
gbev.fit(w,y,
weights=NULL,
measErrorModel=NULL,
method="L2",
indepFitUpdate=1,
nboost=1000,
lambda=100,
maxDepth=2,
m=1,
minSplit=10,
minBucket=0,
sPoints=10,
mc=2,
intermPred=10,
maxSplitAttempts=10)
```

Arguments

w	Matrix of covariates.
y	A vector of responses.
weights	Weights applied to observations. Defaults to 1.

measErrorModel	This is a list specifying the distribution of the latent covariates and the measurement error. Here it is assumed that the latent covariates are a mixture of normals (possibly multivariate), and that the measurement error is normally distributed. See examples below for details.
method	Can be L2 for squared error loss, logLike for binary regression with negative log-likelihood loss.
indepFitUpdate	If indepFitUpdate=1 then model fit is updated using independent MC-sample, else the fit is updated using the same MC-sample as used in tree fitting.
nboost	Number of boosting iterations to perform.
lambda	Regularization parameter.
maxDepth	Determines maximum interaction depth of trees fit. maxDepth=2 fits stumps.
m	Number of randomly sampled covariates used to split node, usually set to number of covariates.
minSplit	Minimum expected number of observations in a node for it to be split.
minBucket	Minimum expected number of observations in a node.
sPoints	Number of points sampled at random from which to choose split.
mc	Number of Monte-Carlo samples drawn to compute node probabilities.
intermPred	Increments of iterations at which intermediate predictions are saved, required for cvLoss function.
maxSplitAttempts	Maximum number of attempts at finding a valid split point. When splitting a node, sPoints candidate splits are supposed to be found for each covariate, however, each randomly sampled split point does not necessarily give a valid split point (i.e. a point satisfying minBucket and minSplit), and maxSplitAttempts is the maximum number of attempts at finding such a point. .

Details

This function is called by gbev but is not intended for end-user use.

mixModelMatrices *Helper function for gbev-function.*

Description

A help function for creating measurement error model, not intended for end-user use.

Usage

```
mixModelMatrices(w, numComp, mu, SigmaME, SigmaJ, pComp)
```

Arguments

w	A matrix of covariates.
numComp	Number of components in mixture of normals model.
mu	Mean vectors of mixture of normal model.
SigmaME	Covariance matrix of measurement error.
SigmaJ	Covariance matrices of mixture of normal model.
pComp	Vector of weights of mixture.

part.dep

Partial dependence

Description

Partial dependence function for boosted tree model.

Usage

```
part.dep(object, varIndx, ngrid=50,
         firstTree=NULL, lastTree=NULL, plt=TRUE)
```

Arguments

object	A fitted object of type gbev.object.
varIndx	Index of variables to compute partial dependence, can be of length 1 or 2, corresponding univariate and bivariate partial dependence functions.
ngrid	Number of grid points along each axis at which to compute partial dependence function.
firstTree	Index of tree in boosted tree sequence to start computations, defaults to 1.
lastTree	Index of last tree in boosted tree sequence to use for computations, defaults to number of boosting iterations.
plt	If TRUE then the partial dependence is plotted.

Details

Computes partial dependence functions used to summarize the marginal effect of covariates on the response. These were introduced in Friedman (2001) and are also described in Hastie et al. (2001) in chapter 10.13.2. The part.dep function, however, computes the partial dependence of the latent covariates, the X 's, on the response, and not the error-contaminated W 's. This requires a slight modification of the procedure described in the references. Specifically, Friedman (2001) describes a procedure where to compute the partial dependence using boosted tree models one needs to know the proportion of the observations falling in the various terminal nodes of the trees. With measurement error, however, this proportion is not observable and must be estimated, which here is done using the Monte Carlo samples used in the tree fitting of each boosting iteration.

Value

A list with elements `pred`, `x` and `dat`. If univariate partial dependence, `pred` is the partial dependence at the points `x`. If bivariate partial dependence, then `dat` is a data-frame with the partial dependence given in variable `z` evaluated at the variables `x` and `y`.

References

J.H. Friedman (2001). "Greedy Function Approximation: A Gradient Boosting Machine," *Annals of Statistics* 29(5):1189-1232.

T. Hastie, R. Tibshirani and J.H. Friedman (2001). "The Elements of Statistical Learning" Springer.

plotLoss

Plot loss function versus iteration.

Description

The loss function at various boosting iterations is plotted.

Usage

```
plotLoss(object, loss="L2", startIter=1, plt=TRUE)
```

Arguments

<code>object</code>	A fitted object of type <code>gbev</code> . object.
<code>loss</code>	Can be "L2" or "logLike" for binary regression.
<code>startIter</code>	Iteration at which to start plotting.
<code>plt</code>	If TRUE then the loss is plotted, otherwise it is only returned.

Details

The loss function at increments of iterations equal to `object$intermPred` is plotted and returned.

Value

The function returns `iters` which are the iterations at which loss is evaluated, and `loss` the loss function value.

predict_gbev	<i>Predictions from boosted tree model.</i>
--------------	---

Description

Returns value of fitted function at given points.

Usage

```
predict_gbev(object, newdata, latent=TRUE, firstTree=NULL, lastTree=NULL)
```

Arguments

object	A fitted object of type gbev.object.
newdata	A data frame from which predictions are made.
latent	If TRUE the value of the fitted function between the error-free (but unobserved) covariates and the response is returned, if FALSE nothing happens.
firstTree	Index of first tree from which predictions are computed. Defaults to 1.
lastTree	Index of last tree from which predictions are computed. Defaults to number of trees in fitting.

Value

pred the predictions/function evaluations.

treeMatrix	<i>Matrix with boosted trees.</i>
------------	-----------------------------------

Description

A matrix containing all of the boosted trees is returned.

Usage

```
treeMatrix(object, deci=3)
```

Arguments

object	A fitted object of type gbev.object.
deci	number of decimal points of reals.

Value

Returns a matrix with all the trees.

var.imp	<i>Variable importance.</i>
---------	-----------------------------

Description

Computes and plots the variable importance of variables used in model.

Usage

```
var.imp(object, nboosts=NULL, std=FALSE, plt=TRUE)
```

Arguments

object	A fitted object of type <code>gbev.object</code> .
nboosts	Number of boosting iterations, if <code>NULL</code> set to maximum number used.
std	If <code>FALSE</code> then the reduction in mean squared error due to covariates is given, if <code>TRUE</code> then this reduction is standardized with respect to variance of response.
plt	If <code>TRUE</code> the importances are plotted.

Details

Variable importance is defined as the reduction in mean squared error due to the different covariates. For a given covariate, this is computed by identifying all nodes split using that covariate and adding together the reductions in mean squared error due to splitting these nodes.

Index

*Topic **hplot**

- checkMeasErrorModel, 2
- cvLoss, 2
- mixModelMatrices, 9
- part.dep, 10
- plotLoss, 11
- predict_gbev, 12
- treeMatrix, 12
- var.imp, 13

*Topic **nonparametric**

- gbev, 3
- gbev.fit, 8

*Topic **tree**

- gbev, 3
- gbev.fit, 8

checkMeasErrorModel, 2

cvLoss, 2

gbev, 3

gbev.fit, 8

mixModelMatrices, 9

part.dep, 10

plotLoss, 11

predict_gbev, 12

treeMatrix, 12

var.imp, 13