Package 'gbm'

March 11, 2015

Version 2.1.1

Date 2015-03-10

Title Generalized Boosted Regression Models

Author Greg Ridgeway <gregridgeway@gmail.com> with contributions from others

Maintainer Harry Southworth <harry.southworth@gmail.com>

Depends R (>= 2.9.0), survival, lattice, splines, parallel

Suggests RUnit

Description An implementation of extensions to Freund and Schapire's AdaBoost algorithm and Friedman's gradient boosting machine. Includes regression methods for least squares, absolute loss, t-distribution loss, quantile regression, logistic, multinomial logistic, Poisson, Cox proportional hazards partial likelihood, AdaBoost exponential loss, Huberized hinge loss, and Learning to Rank measures (LambdaMart).

License GPL (>= 2) | file LICENSE

URL <http://code.google.com/p/gradientboostedmodels/>

NeedsCompilation yes

Repository CRAN

Date/Publication 2015-03-11 09:02:32

R topics documented:

---

gbm-package   *Generalized Boosted Regression Models*

Description

This package implements extensions to Freund and Schapire's AdaBoost algorithm and J. Friedman's gradient boosting machine. Includes regression methods for least squares, absolute loss, logistic, Poisson, Cox proportional hazards partial likelihood, multinomial, t-distribution, AdaBoost exponential loss, Learning to Rank, and Huberized hinge loss.

Details

Package:	gbm
Version:	2.1
Date:	2013-05-10
Depends:	R (>= 2.9.0), survival, lattice, mgcv
License:	GPL (version 2 or newer)
URL:	http://code.google.com/p/gradientboostedmodels/

Index:

```
basehaz.gbm          Baseline hazard function
calibrate.plot       Calibration plot
gbm                  Generalized Boosted Regression Modeling
gbm.object          Generalized Boosted Regression Model Object
gbm.perf            GBM performance
plot.gbm            Marginal plots of fitted gbm objects
predict.gbm         Predict method for GBM Model Fits
pretty.gbm.tree     Print gbm tree components
```

```
quantile.rug          Quantile rug plot
relative.influence    Methods for estimating relative influence
shrink.gbm            L1 shrinkage of the predictor variables in a GBM
shrink.gbm.pred       Predictions from a shrunked GBM
summary.gbm           Summary of a gbm object
```

Further information is available in the following vignettes:

gbm   Generalized Boosted Models: A guide to the gbm package (source, pdf)

Author(s)

Greg Ridgeway <gregridgeway@gmail.com> with contributions by Daniel Edwards, Brian Kriegler, Stefan Schroedl and Harry Southworth.

References

Y. Freund and R.E. Schapire (1997) "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences,* 55(1):119-139.

G. Ridgeway (1999). "The state of boosting," *Computing Science and Statistics* 31:172-181.

J.H. Friedman, T. Hastie, R. Tibshirani (2000). "Additive Logistic Regression: a Statistical View of Boosting," *Annals of Statistics* 28(2):337-374.

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

J.H. Friedman (2002). "Stochastic Gradient Boosting," *Computational Statistics and Data Analysis* 38(4):367-378.

The MART website.

---

basehaz.gbm   *Baseline hazard function*

Description

Computes the Breslow estimator of the baseline hazard function for a proportional hazard regression model

Usage

```
basehaz.gbm(t, delta, f.x,
            t.eval = NULL,
            smooth = FALSE,
            cumulative = TRUE)
```

Arguments

t	the survival times
delta	the censoring indicator
f.x	the predicted values of the regression model on the log hazard scale
t.eval	values at which the baseline hazard will be evaluated
smooth	if TRUE basehaz.gbm will smooth the estimated baseline hazard using Friedman's super smoother [supsmu](#)
cumulative	if TRUE the cumulative survival function will be computed

Details

The proportional hazard model assumes h(t|x)=lambda(t)*exp(f(x)). [gbm](#) can estimate the f(x) component via partial likelihood. After estimating f(x), basehaz.gbm can compute the a nonparametric estimate of lambda(t).

Value

a vector of length equal to the length of t (or of length t.eval if t.eval is not NULL) containing the baseline hazard evaluated at t (or at t.eval if t.eval is not NULL). If cumulative is set to TRUE then the returned vector evaluates the cumulative hazard function at those values.

Author(s)

Greg Ridgeway <gregridgeway@gmail.com>

References

N. Breslow (1972). "Disussion of 'Regression Models and Life-Tables' by D.R. Cox," Journal of the Royal Statistical Society, Series B, 34(2):216-217.

N. Breslow (1974). "Covariance analysis of censored survival data," Biometrics 30:89-99.

See Also

[survfit](#), [gbm](#)

---

calibrate.plot   *Calibration plot*

Description

An experimental diagnostic tool that plots the fitted values versus the actual average values. Currently developed for only distribution="bernoulli".

Usage

```
calibrate.plot(y,p,
               distribution="bernoulli",
               replace=TRUE,
               line.par=list(col="black"),
               shade.col="lightyellow",
               shade.density=NULL,
               rug.par=list(side=1),
               xlab="Predicted value",
               ylab="Observed average",
               xlim=NULL,ylim=NULL,
               knots=NULL,df=6,
               ...)
```

Arguments

y	the outcome 0-1 variable
p	the predictions estimating E(y|x)
distribution	the loss function used in creating p. bernoulli and poisson are currently the only special options. All others default to squared error assuming gaussian
replace	determines whether this plot will replace or overlay the current plot. replace=FALSE is useful for comparing the calibration of several methods
line.par	graphics parameters for the line
shade.col	color for shading the 2 SE region. shade.col=NA implies no 2 SE region
shade.density	the density parameter for [polygon](#)
rug.par	graphics parameters passed to [rug](#)
xlab	x-axis label corresponding to the predicted values
ylab	y-axis label corresponding to the observed average
xlim,ylim	x and y-axis limits. If not specified te function will select limits
knots,df	these parameters are passed directly to [ns](#) for constructing a natural spline smoother for the calibration curve
...	other graphics parameters passed on to the plot function

Details

Uses natural splines to estimate E(y|p). Well-calibrated predictions imply that E(y|p) = p. The plot also includes a pointwise 95 band.

Value

calibrate.plot returns no values.

Author(s)

Greg Ridgeway <gregridgeway@gmail.com>

References

J.F. Yates (1982). "External correspondence: decomposition of the mean probability score," Organisational Behaviour and Human Performance 30:132-156.

D.J. Spiegelhalter (1986). "Probabilistic Prediction in Patient Management and Clinical Trials," Statistics in Medicine 5:421-433.

Examples

```
# Don't want R CMD check to think there is a dependency on rpart
# so comment out the example
#library(rpart)
#data(kyphosis)
#y <- as.numeric(kyphosis$Kyphosis)-1
#x <- kyphosis$Age
#glm1 <- glm(y~poly(x,2),family=binomial)
#p <- predict(glm1,type="response")
#calibrate.plot(y, p, xlim=c(0,0.6), ylim=c(0,0.6))
```

---

gbm	*Generalized Boosted Regression Modeling*

Description

Fits generalized boosted regression models.

Usage

```
gbm(formula = formula(data),
    distribution = "bernoulli",
    data = list(),
    weights,
    var.monotone = NULL,
    n.trees = 100,
    interaction.depth = 1,
    n.minobsinnode = 10,
    shrinkage = 0.001,
    bag.fraction = 0.5,
    train.fraction = 1.0,
    cv.folds=0,
    keep.data = TRUE,
    verbose = "CV",
    class.stratify.cv=NULL,
    n.cores = NULL)

gbm.fit(x, y,
        offset = NULL,
        misc = NULL,
```

```
          distribution = "bernoulli",
          w = NULL,
          var.monotone = NULL,
          n.trees = 100,
          interaction.depth = 1,
          n.minobsinnode = 10,
          shrinkage = 0.001,
          bag.fraction = 0.5,
          nTrain = NULL,
          train.fraction = NULL,
          keep.data = TRUE,
          verbose = TRUE,
          var.names = NULL,
          response.name = "y",
          group = NULL)

gbm.more(object,
          n.new.trees = 100,
          data = NULL,
          weights = NULL,
          offset = NULL,
          verbose = NULL)
```

Arguments

formula   a symbolic description of the model to be fit. The formula may include an offset
                 term (e.g. y~offset(n)+x). If keep.data=FALSE in the initial call to gbm then it
                 is the user's responsibility to resupply the offset to gbm.more.

distribution   either a character string specifying the name of the distribution to use or a list
                 with a component name specifying the distribution and any additional param-
                 eters needed. If not specified, gbm will try to guess: if the response has only
                 2 unique values, bernoulli is assumed; otherwise, if the response is a factor,
                 multinomial is assumed; otherwise, if the response has class "Surv", coxph is
                 assumed; otherwise, gaussian is assumed.

                 Currently available options are "gaussian" (squared error), "laplace" (absolute
                 loss), "tdist" (t-distribution loss), "bernoulli" (logistic regression for 0-1 out-
                 comes), "huberized" (huberized hinge loss for 0-1 outcomes), "multinomial"
                 (classification when there are more than 2 classes), "adaboost" (the AdaBoost
                 exponential loss for 0-1 outcomes), "poisson" (count outcomes), "coxph" (right
                 censored observations), "quantile", or "pairwise" (ranking measure using the
                 LambdaMart algorithm).

                 If quantile regression is specified, distribution must be a list of the form
                 list(name="quantile",alpha=0.25) where alpha is the quantile to estimate.
                 The current version's quantile regression method does not handle non-constant
                 weights and will stop.

                 If "tdist" is specified, the default degrees of freedom is 4 and this can be con-
                 trolled by specifying distribution=list(name="tdist", df=DF) where DF
                 is your chosen degrees of freedom.
```

If "pairwise" regression is specified, distribution must be a list of the form `list(name="pairwise",group=...,metric=...,max.rank=...)` (metric and max.rank are optional, see below). group is a character vector with the column names of data that jointly indicate the group an instance belongs to (typically a query in Information Retrieval applications). For training, only pairs of instances from the same group and with different target labels can be considered. metric is the IR measure to use, one of

conc: Fraction of concordant pairs; for binary labels, this is equivalent to the Area under the ROC Curve

mrr: Mean reciprocal rank of the highest-ranked positive instance

map: Mean average precision, a generalization of mrr to multiple positive instances

ndcg: Normalized discounted cumulative gain. The score is the weighted sum (DCG) of the user-supplied target values, weighted by log(rank+1), and normalized to the maximum achievable value. This is the default if the user did not specify a metric.

ndcg and conc allow arbitrary target values, while binary targets {0,1} are expected for map and mrr. For ndcg and mrr, a cut-off can be chosen using a positive integer parameter max.rank. If left unspecified, all ranks are taken into account.

Note that splitting of instances into training and validation sets follows group boundaries and therefore only approximates the specified train.fraction ratio (the same applies to cross-validation folds). Internally, queries are randomly shuffled before training, to avoid bias.

Weights can be used in conjunction with pairwise metrics, however it is assumed that they are constant for instances from the same group.

For details and background on the algorithm, see e.g. Burges (2010).

data            an optional data frame containing the variables in the model. By default the variables are taken from environment(formula), typically the environment from which gbm is called. If keep.data=TRUE in the initial call to gbm then gbm stores a copy with the object. If keep.data=FALSE then subsequent calls to gbm.more must resupply the same dataset. It becomes the user's responsibility to resupply the same data at this point.

weights         an optional vector of weights to be used in the fitting process. Must be positive but do not need to be normalized. If keep.data=FALSE in the initial call to gbm then it is the user's responsibility to resupply the weights to gbm.more.

var.monotone    an optional vector, the same length as the number of predictors, indicating which variables have a monotone increasing (+1), decreasing (-1), or arbitrary (0) relationship with the outcome.

n.trees         the total number of trees to fit. This is equivalent to the number of iterations and the number of basis functions in the additive expansion.

cv.folds        Number of cross-validation folds to perform. If cv.folds>1 then gbm, in addition to the usual fit, will perform a cross-validation, calculate an estimate of generalization error returned in cv.error.

interaction.depth
: The maximum depth of variable interactions. 1 implies an additive model, 2 implies a model with up to 2-way interactions, etc.

n.minobsinnode
: minimum number of observations in the trees terminal nodes. Note that this is the actual number of observations not the total weight.

shrinkage
: a shrinkage parameter applied to each tree in the expansion. Also known as the learning rate or step-size reduction.

bag.fraction
: the fraction of the training set observations randomly selected to propose the next tree in the expansion. This introduces randomnesses into the model fit. If bag.fraction<1 then running the same model twice will result in similar but different fits. gbm uses the R random number generator so set.seed can ensure that the model can be reconstructed. Preferably, the user can save the returned [gbm.object](#) using [save](#).

train.fraction
: The first train.fraction * nrows(data) observations are used to fit the gbm and the remainder are used for computing out-of-sample estimates of the loss function.

nTrain
: An integer representing the number of cases on which to train. This is the preferred way of specification for gbm.fit; The option train.fraction in gbm.fit is deprecated and only maintained for backward compatibility. These two parameters are mutually exclusive. If both are unspecified, all data is used for training.

keep.data
: a logical variable indicating whether to keep the data and an index of the data stored with the object. Keeping the data and index makes subsequent calls to [gbm.more](#) faster at the cost of storing an extra copy of the dataset.

object
: a gbm object created from an initial call to [gbm](#).

n.new.trees
: the number of additional trees to add to object.

verbose
: If TRUE, gbm will print out progress and performance indicators. If this option is left unspecified for gbm.more then it uses verbose from object.

class.stratify.cv
: whether or not the cross-validation should be stratified by class. Defaults to TRUE for distribution="multinomial" and is only implemented for multinomial and bernoulli. The purpose of stratifying the cross-validation is to help avoiding situations in which training sets do not contain all classes.

x, y
: For gbm.fit: x is a data frame or data matrix containing the predictor variables and y is the vector of outcomes. The number of rows in x must be the same as the length of y.

offset
: a vector of values for the offset

misc
: For gbm.fit: misc is an R object that is simply passed on to the gbm engine. It can be used for additional data for the specific distribution. Currently it is only used for passing the censoring indicator for the Cox proportional hazards model.

w
: For gbm.fit: w is a vector of weights of the same length as the y.

var.names
: For gbm.fit: A vector of strings of length equal to the number of columns of x containing the names of the predictor variables.

response.name
: For gbm.fit: A character string label for the response variable.

| | |
|---|---|
| group | group used when distribution = 'pairwise'. |
| n.cores | The number of CPU cores to use. The cross-validation loop will attempt to send different CV folds off to different cores. If n.cores is not specified by the user, it is guessed using the detectCores function in the parallel package. Note that the documentation for detectCores makes clear that it is not failsave and could return a spurious number of available cores. |

### Details

See the gbm vignette for technical details.

This package implements the generalized boosted modeling framework. Boosting is the process of iteratively adding basis functions in a greedy fashion so that each additional basis function further reduces the selected loss function. This implementation closely follows Friedman's Gradient Boosting Machine (Friedman, 2001).

In addition to many of the features documented in the Gradient Boosting Machine, gbm offers additional features including the out-of-bag estimator for the optimal number of iterations, the ability to store and manipulate the resulting gbm object, and a variety of other loss functions that had not previously had associated boosting algorithms, including the Cox partial likelihood for censored data, the poisson likelihood for count outcomes, and a gradient boosting implementation to minimize the AdaBoost exponential loss function.

gbm.fit provides the link between R and the C++ gbm engine. gbm is a front-end to gbm.fit that uses the familiar R modeling formulas. However, model.frame is very slow if there are many predictor variables. For power-users with many variables use gbm.fit. For general practice gbm is preferable.

### Value

gbm, gbm.fit, and gbm.more return a gbm.object.

### Author(s)

Greg Ridgeway <gregridgeway@gmail.com>

Quantile regression code developed by Brian Kriegler <bk@stat.ucla.edu>

t-distribution, and multinomial code developed by Harry Southworth and Daniel Edwards

Pairwise code developed by Stefan Schroedl <schroedl@a9.com>

### References

Y. Freund and R.E. Schapire (1997) "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences,* 55(1):119-139.

G. Ridgeway (1999). "The state of boosting," *Computing Science and Statistics* 31:172-181.

J.H. Friedman, T. Hastie, R. Tibshirani (2000). "Additive Logistic Regression: a Statistical View of Boosting," *Annals of Statistics* 28(2):337-374.

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

J.H. Friedman (2002). "Stochastic Gradient Boosting," *Computational Statistics and Data Analysis* 38(4):367-378.

B. Kriegler (2007). Cost-Sensitive Stochastic Gradient Boosting Within a Quantitative Regression Framework. PhD dissertation, UCLA Statistics.

C. Burges (2010). "From RankNet to LambdaRank to LambdaMART: An Overview," Microsoft Research Technical Report MSR-TR-2010-82.

Greg Ridgeway's site.

The MART website.

## See Also

gbm.object, gbm.perf, plot.gbm, predict.gbm, summary.gbm, pretty.gbm.tree.

## Examples

```
 # A least squares regression example # create some data

N <- 1000
X1 <- runif(N)
X2 <- 2*runif(N)
X3 <- ordered(sample(letters[1:4],N,replace=TRUE),levels=letters[4:1])
X4 <- factor(sample(letters[1:6],N,replace=TRUE))
X5 <- factor(sample(letters[1:3],N,replace=TRUE))
X6 <- 3*runif(N)
mu <- c(-1,0,1,2)[as.numeric(X3)]

SNR <- 10 # signal-to-noise ratio
Y <- X1**1.5 + 2 * (X2**.5) + mu
sigma <- sqrt(var(Y)/SNR)
Y <- Y + rnorm(N,0,sigma)

# introduce some missing values
X1[sample(1:N,size=500)] <- NA
X4[sample(1:N,size=300)] <- NA

data <- data.frame(Y=Y,X1=X1,X2=X2,X3=X3,X4=X4,X5=X5,X6=X6)

# fit initial model
gbm1 <-
gbm(Y~X1+X2+X3+X4+X5+X6,         # formula
    data=data,                   # dataset
    var.monotone=c(0,0,0,0,0,0), # -1: monotone decrease,
                                 # +1: monotone increase,
                                 #  0: no monotone restrictions
    distribution="gaussian",     # see the help for other choices
    n.trees=1000,                # number of trees
    shrinkage=0.05,              # shrinkage or learning rate,
                                 # 0.001 to 0.1 usually work
    interaction.depth=3,         # 1: additive model, 2: two-way interactions, etc.
    bag.fraction = 0.5,          # subsampling fraction, 0.5 is probably best
    train.fraction = 0.5,        # fraction of data for training,
```

```
                                   # first train.fraction*N used for training
    n.minobsinnode = 10,           # minimum total weight needed in each node
    cv.folds = 3,                  # do 3-fold cross-validation
    keep.data=TRUE,                # keep a copy of the dataset with the object
    verbose=FALSE,                 # don't print out progress
    n.cores=1)                     # use only a single core (detecting #cores is
                                   # error-prone, so avoided here)

# check performance using an out-of-bag estimator
# OOB underestimates the optimal number of iterations
best.iter <- gbm.perf(gbm1,method="OOB")
print(best.iter)

# check performance using a 50% heldout test set
best.iter <- gbm.perf(gbm1,method="test")
print(best.iter)

# check performance using 5-fold cross-validation
best.iter <- gbm.perf(gbm1,method="cv")
print(best.iter)

# plot the performance # plot variable influence
summary(gbm1,n.trees=1)         # based on the first tree
summary(gbm1,n.trees=best.iter) # based on the estimated best number of trees

# compactly print the first and last trees for curiosity
print(pretty.gbm.tree(gbm1,1))
print(pretty.gbm.tree(gbm1,gbm1$n.trees))

# make some new data
N <- 1000
X1 <- runif(N)
X2 <- 2*runif(N)
X3 <- ordered(sample(letters[1:4],N,replace=TRUE))
X4 <- factor(sample(letters[1:6],N,replace=TRUE))
X5 <- factor(sample(letters[1:3],N,replace=TRUE))
X6 <- 3*runif(N)
mu <- c(-1,0,1,2)[as.numeric(X3)]

Y <- X1**1.5 + 2 * (X2**.5) + mu + rnorm(N,0,sigma)

data2 <- data.frame(Y=Y,X1=X1,X2=X2,X3=X3,X4=X4,X5=X5,X6=X6)

# predict on the new data using "best" number of trees
# f.predict generally will be on the canonical scale (logit,log,etc.)
f.predict <- predict(gbm1,data2,best.iter)

# least squares error
print(sum((data2$Y-f.predict)^2))

# create marginal plots
# plot variable X1,X2,X3 after "best" iterations
par(mfrow=c(1,3))
```

```
plot(gbm1,1,best.iter)
plot(gbm1,2,best.iter)
plot(gbm1,3,best.iter)
par(mfrow=c(1,1))
# contour plot of variables 1 and 2 after "best" iterations
plot(gbm1,1:2,best.iter)
# lattice plot of variables 2 and 3
plot(gbm1,2:3,best.iter)
# lattice plot of variables 3 and 4
plot(gbm1,3:4,best.iter)

# 3-way plots
plot(gbm1,c(1,2,6),best.iter,cont=20)
plot(gbm1,1:3,best.iter)
plot(gbm1,2:4,best.iter)
plot(gbm1,3:5,best.iter)

# do another 100 iterations
gbm2 <- gbm.more(gbm1,100,
                 verbose=FALSE) # stop printing detailed progress
```

---

gbm-internal   *gbm internal functions*

Description

Helper functions for preprocessing data prior to building the model

Usage

```
guessDist(y)
getCVgroup(distribution, class.stratify.cv, y, i.train, cv.folds, group)
getStratify(strat, d)
checkMissing(x, y)
checkWeights(w, n)
checkID(id)
checkOffset(o, y)
getVarNames(x)
gbmCluster(n)
```

Arguments

y	The response variable
d, distribution	
	The distribution, either specified by the user or implied
class.stratify.cv	
	Whether or not to stratify, if provided by the user
i.train	Computed internally by gbm

group	The group, if using distibution='pairwise'
strat	Whether or not to stratify
cv.folds	The number of cross-validation folds
x	The design matrix
id	The interaction depth
w	The weights
n	The number of cores to use in the cluster.
o	The offset

Details

These are functions used internally by gbm and not intended for direct use by the user.

---

gbm.object	*Generalized Boosted Regression Model Object*

Description

These are objects representing fitted gbms.

Value

initF	the "intercept" term, the initial predicted value to which trees make adjustments
fit	a vector containing the fitted values on the scale of regression function (e.g. log-odds scale for bernoulli, log scale for poisson)
train.error	a vector of length equal to the number of fitted trees containing the value of the loss function for each boosting iteration evaluated on the training data
valid.error	a vector of length equal to the number of fitted trees containing the value of the loss function for each boosting iteration evaluated on the validation data
cv.error	if cv.folds<2 this component is NULL. Otherwise, this component is a vector of length equal to the number of fitted trees containing a cross-validated estimate of the loss function for each boosting iteration
oobag.improve	a vector of length equal to the number of fitted trees containing an out-of-bag estimate of the marginal reduction in the expected value of the loss function. The out-of-bag estimate uses only the training data and is useful for estimating the optimal number of boosting iterations. See gbm.perf
trees	a list containing the tree structures. The components are best viewed using pretty.gbm.tree
c.splits	a list of all the categorical splits in the collection of trees. If the trees[[i]] component of a gbm object describes a categorical split then the splitting value will refer to a component of c.splits. That component of c.splits will be a vector of length equal to the number of levels in the categorical split variable. -1 indicates left, +1 indicates right, and 0 indicates that the level was not present in the training data

cv.fitted   If cross-validation was performed, the cross-validation predicted values on the scale of the linear predictor. That is, the fitted values from the ith CV-fold, for the model having been trained on the data in all other folds.

Structure

The following components must be included in a legitimate gbm object.

Author(s)

Greg Ridgeway <gregridgeway@gmail.com>

See Also

gbm

---

gbm.perf *GBM performance*

Description

Estimates the optimal number of boosting iterations for a gbm object and optionally plots various performance measures

Usage

```
gbm.perf(object,
         plot.it = TRUE,
         oobag.curve = FALSE,
         overlay = TRUE,
         method)
```

Arguments

object   a gbm.object created from an initial call to gbm.

plot.it   an indicator of whether or not to plot the performance measures. Setting plot.it=TRUE creates two plots. The first plot plots object$train.error (in black) and object$valid.error (in red) versus the iteration number. The scale of the error measurement, shown on the left vertical axis, depends on the distribution argument used in the initial call to gbm.

oobag.curve   indicates whether to plot the out-of-bag performance measures in a second plot.

overlay   if TRUE and oobag.curve=TRUE then a right y-axis is added to the training and test error plot and the estimated cumulative improvement in the loss function is plotted versus the iteration number.

method   indicate the method used to estimate the optimal number of boosting iterations. method="OOB" computes the out-of-bag estimate and method="test" uses the test (or validation) dataset to compute an out-of-sample estimate. method="cv" extracts the optimal number of iterations using cross-validation if gbm was called with cv.folds>1

Value

`gbm.perf` returns the estimated optimal number of iterations. The method of computation depends on the `method` argument.

Author(s)

Greg Ridgeway <gregridgeway@gmail.com>

See Also

[gbm](), [gbm.object]()

---

gbm.roc.area   *Compute Information Retrieval measures.*

Description

Functions to compute Information Retrieval measures for pairwise loss for a single group. The function returns the respective metric, or a negative value if it is undefined for the given group.

Usage

```
gbm.roc.area(obs, pred)
ir.measure.conc(y.f, max.rank)
ir.measure.auc(y.f, max.rank)
ir.measure.mrr(y.f, max.rank)
ir.measure.map(y.f, max.rank)
ir.measure.ndcg(y.f, max.rank)
perf.pairwise(y, f, group, metric="ndcg", w=NULL, max.rank=0)
```

Arguments

obs	Observed value
pred	Predicted value
metric	What type of performance measure to compute.
y, y.f, f, w, group, max.rank	
	Used internally.

Details

For simplicity, we have no special handling for ties; instead, we break ties randomly. This is slightly inaccurate for individual groups, but should have only a small effect on the overall measure.

`gbm.conc` computes the concordance index: Fraction of all pairs (i,j) with i<j, x[i] != x[j], such that x[j] < x[i]

If `obs` is binary, then `gbm.roc.area(obs, pred) = gbm.conc(obs[order(-pred)])`.

`gbm.conc` is more general as it allows non-binary targets, but is significantly slower.

Value

The requested performance measure.

Author(s)

Stefan Schroedl

References

C. Burges (2010). "From RankNet to LambdaRank to LambdaMART: An Overview", Microsoft Research Technical Report MSR-TR-2010-82.

See Also

[gbm](gbm)

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.
```

---

gbmCrossVal   *Cross-validate a gbm*

Description

Functions for cross-validating gbm. These functions are used internally and are not intended for end-user direct usage.

Usage

```
gbmCrossVal(cv.folds, nTrain, n.cores,
class.stratify.cv, data,
x, y, offset, distribution, w, var.monotone,
n.trees, interaction.depth, n.minobsinnode,
shrinkage, bag.fraction,
var.names, response.name, group)

gbmCrossValModelBuild(cv.folds, cv.group, n.cores,
i.train, x, y, offset,
distribution, w, var.monotone,
n.trees, interaction.depth,
n.minobsinnode, shrinkage,
bag.fraction, var.names,
response.name, group)
```

```
gbmDoFold(X, i.train, x, y, offset, distribution, w, var.monotone, n.trees,
          interaction.depth, n.minobsinnode, shrinkage, bag.fraction,
          cv.group, var.names, response.name, group, s)

gbmCrossValErr(cv.models, cv.folds, cv.group, nTrain, n.trees)

gbmCrossValPredictions(cv.models, cv.folds, cv.group,
best.iter.cv, distribution, data, y)
```

Arguments

cv.folds	The number of cross-validation folds.
nTrain	The number of training samples.
n.cores	The number of cores to use.
class.stratify.cv	
	Whether or not stratified cross-validation samples are used.
data	The data.
x	The model matrix.
y	The response variable.
offset	The offset.
distribution	The type of loss function. See [gbm](#).
w	Observation weights.
var.monotone	See [gbm](#).
n.trees	The number of trees to fit.
interaction.depth	
	The degree of allowed interactions. See [gbm](#).
n.minobsinnode	See [gbm](#).
shrinkage	See [gbm](#).
bag.fraction	See [gbm](#).
var.names	See [gbm](#).
response.name	See [gbm](#).
group	Used when distribution = "pairwise". See [gbm](#).
i.train	Items in the training set.
cv.models	A list containing the models for each fold.
cv.group	A vector indicating the cross-validation fold for each member of the training set.
best.iter.cv	The iteration with lowest cross-validation error.
X	Index (cross-validation fold) on which to subset.
s	Random seed.

Details

These functions are not intended for end-user direct usage, but are used internally by gbm.

Value

A list containing the cross-validation error and predictions.

Author(s)

Greg Ridgeway <gregridgeway@gmail.com>

References

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

L. Breiman (2001). Random Forests.

See Also

gbm

---

interact.gbm   *Estimate the strength of interaction effects*

Description

Computes Friedman's H-statistic to assess the strength of variable interactions.

Usage

```
interact.gbm(x,
             data,
             i.var = 1,
             n.trees = x$n.trees)
```

Arguments

x	a gbm.object fitted using a call to gbm
data	the dataset used to construct x. If the original dataset is large, a random subsample may be used to accelerate the computation in interact.gbm
i.var	a vector of indices or the names of the variables for compute the interaction effect. If using indices, the variables are indexed in the same order that they appear in the initial gbm formula.
n.trees	the number of trees used to generate the plot. Only the first n.trees trees will be used

Details

`interact.gbm` computes Friedman's H-statistic to assess the relative strength of interaction effects in non-linear models. H is on the scale of [0-1] with higher values indicating larger interaction effects. To connect to a more familiar measure, if x_1 and x_2 are uncorrelated covariates with mean 0 and variance 1 and the model is of the form

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$

then

$$H = \frac{\beta_3}{\sqrt{\beta_1^2 + \beta_2^2 + \beta_3^2}}$$

Note that if the main effects are weak, the estimated H will be unstable. For example, if (in the case of a two-way interaction) neither main effect is in the selected model (relative influence is zero), the result will be 0/0. Also, with weak main effects, rounding errors can result in values of H > 1 which are not possible.

Value

Returns the value of H.

Author(s)

Greg Ridgeway <gregridgeway@gmail.com>

References

J.H. Friedman and B.E. Popescu (2005). "Predictive Learning via Rule Ensembles." Section 8.1

See Also

gbm, gbm.object

---

plot.gbm   *Marginal plots of fitted gbm objects*

Description

Plots the marginal effect of the selected variables by "integrating" out the other variables.

Usage

```
## S3 method for class 'gbm'
plot(x,
     i.var = 1,
     n.trees = x$n.trees,
     continuous.resolution = 100,
     return.grid = FALSE,
     type = "link",
     ...)
```

Arguments

x	a gbm.object fitted using a call to gbm
i.var	a vector of indices or the names of the variables to plot. If using indices, the variables are indexed in the same order that they appear in the initial gbm formula. If length(i.var) is between 1 and 3 then plot.gbm produces the plots. Otherwise, plot.gbm returns only the grid of evaluation points and their average predictions
n.trees	the number of trees used to generate the plot. Only the first n.trees trees will be used
continuous.resolution	
	The number of equally space points at which to evaluate continuous predictors
return.grid	if TRUE then plot.gbm produces no graphics and only returns the grid of evaluation points and their average predictions. This is useful for customizing the graphics for special variable types or for dimensions greater than 3
type	the type of prediction to plot on the vertical axis. See predict.gbm
...	other arguments passed to the plot function

Details

plot.gbm produces low dimensional projections of the gbm.object by integrating out the variables not included in the i.var argument. The function selects a grid of points and uses the weighted tree traversal method described in Friedman (2001) to do the integration. Based on the variable types included in the projection, plot.gbm selects an appropriate display choosing amongst line plots, contour plots, and lattice plots. If the default graphics are not sufficient the user may set return.grid=TRUE, store the result of the function, and develop another graphic display more appropriate to the particular example.

Value

Nothing unless return.grid is true then plot.gbm produces no graphics and only returns the grid of evaluation points and their average predictions.

Author(s)

Greg Ridgeway <gregridgeway@gmail.com>

References

J.H. Friedman (2001). "Greedy Function Approximation: A Gradient Boosting Machine," Annals of Statistics 29(4).

See Also

gbm, gbm.object, plot

---

predict.gbm   *Predict method for GBM Model Fits*

Description

Predicted values based on a generalized boosted model object

Usage

```
## S3 method for class 'gbm'
predict(object,
        newdata,
        n.trees,
        type="link",
        single.tree=FALSE,
        ...)
```

Arguments

object	Object of class inheriting from ([gbm.object](#))
newdata	Data frame of observations for which to make predictions
n.trees	Number of trees used in the prediction. n.trees may be a vector in which case predictions are returned for each iteration specified
type	The scale on which gbm makes the predictions
single.tree	If single.tree=TRUE then predict.gbm returns only the predictions from tree(s) n.trees
...	further arguments passed to or from other methods

Details

predict.gbm produces predicted values for each observation in newdata using the the first n.trees iterations of the boosting sequence. If n.trees is a vector than the result is a matrix with each column representing the predictions from gbm models with n.trees[1] iterations, n.trees[2] iterations, and so on.

The predictions from gbm do not include the offset term. The user may add the value of the offset to the predicted value if desired.

If object was fit using [gbm.fit](#) there will be no Terms component. Therefore, the user has greater responsibility to make sure that newdata is of the same format (order and number of variables) as the one originally used to fit the model.

Value

Returns a vector of predictions. By default the predictions are on the scale of f(x). For example, for the Bernoulli loss the returned value is on the log odds scale, poisson loss on the log scale, and coxph is on the log hazard scale.

If `type="response"` then gbm converts back to the same scale as the outcome. Currently the only effect this will have is returning probabilities for bernoulli and expected counts for poisson. For the other distributions "response" and "link" return the same.

Author(s)

Greg Ridgeway <gregridgeway@gmail.com>

See Also

gbm, gbm.object

---

pretty.gbm.tree   *Print gbm tree components*

Description

gbm stores the collection of trees used to construct the model in a compact matrix structure. This function extracts the information from a single tree and displays it in a slightly more readable form. This function is mostly for debugging purposes and to satisfy some users' curiosity.

Usage

```
pretty.gbm.tree(object, i.tree = 1)
```

Arguments

object	a gbm.object initially fit using gbm
i.tree	the index of the tree component to extract from object and display

Value

pretty.gbm.tree returns a data frame. Each row corresponds to a node in the tree. Columns indicate

SplitVar	index of which variable is used to split. -1 indicates a terminal node.
SplitCodePred	if the split variable is continuous then this component is the split point. If the split variable is categorical then this component contains the index of object$c.split that describes the categorical split. If the node is a terminal node then this is the prediction.
LeftNode	the index of the row corresponding to the left node.
RightNode	the index of the row corresponding to the right node.

ErrorReduction	the reduction in the loss function as a result of splitting this node.
Weight	the total weight of observations in the node. If weights are all equal to 1 then this is the number of observations in the node.

Author(s)

Greg Ridgeway <gregridgeway@gmail.com>

See Also

[gbm](#), [gbm.object](#)

---

print.gbm   *Print model summary*

Description

Display basic information about a gbm object.

Usage

```
## S3 method for class 'gbm'
print(x, ...)
## S3 method for class 'gbm'
show(x, ...)
```

Arguments

x	an object of class gbm.
...	arguments passed to print.default.

Details

Prints some information about the model object. In particular, this method prints the call to gbm(), the type of loss function that was used, and the total number of iterations.

If cross-validation was performed, the 'best' number of trees as estimated by cross-validation error is displayed. If a test set was used, the 'best' number of trees as estimated by the test set error is displayed.

The number of available predictors, and the number of those having non-zero influence on predictions is given (which might be interesting in data mining applications).

If multinomial, bernoulli or adaboost was used, the confusion matrix and prediction accuracy are printed (objects being allocated to the class with highest probability for multinomial and bernoulli). These classifications are performed on the entire training data using the model with the 'best' number of trees as described above, or the maximum number of trees if the 'best' cannot be computed.

If the 'distribution' was specified as gaussian, laplace, quantile or t-distribution, a summary of the residuals is displayed. The residuals are for the training data with the model at the 'best' number of trees, as described above, or the maximum number of trees if the 'best' cannot be computed.

Author(s)

Harry Southworth, Daniel Edwards

See Also

[gbm](gbm)

Examples

```
data(iris)
iris.mod <- gbm(Species ~ ., distribution="multinomial", data=iris,
                n.trees=2000, shrinkage=0.01, cv.folds=5,
                verbose=FALSE, n.cores=1)
iris.mod
#data(lung)
#lung.mod <- gbm(Surv(time, status) ~ ., distribution="coxph", data=lung,
#                n.trees=2000, shrinkage=0.01, cv.folds=5,verbose =FALSE)
#lung.mod
```

---

quantile.rug	*Quantile rug plot*

Description

Marks the quantiles on the axes of the current plot.

Usage

```
quantile.rug(x,prob=(0:10)/10,...)
```

Arguments

x	a numeric vector.
prob	the quantiles of x to mark on the x-axis.
...	additional graphics parameters currently ignored.

Value

No return values

Author(s)

Greg Ridgeway <gregridgeway@gmail.com>

See Also

plot, quantile, jitter, [rug](rug).

Examples

```
x <- rnorm(100)
y <- rnorm(100)
plot(x,y)
quantile.rug(x)
```

---

reconstructGBMdata   *Reconstruct a GBM's Source Data*

Description

Helper function to reconstitute the data for plots and summaries. This function is not intended for the user to call directly.

Usage

```
reconstructGBMdata(x)
```

Arguments

x   a `gbm.object` initially fit using `gbm`

Value

Returns a data used to fit the gbm in a format that can subsequently be used for plots and summaries

Author(s)

Harry Southworth

See Also

`gbm`, `gbm.object`

---

relative.influence   *Methods for estimating relative influence*

Description

Helper functions for computing the relative influence of each variable in the gbm object.

Usage

```
relative.influence(object, n.trees, scale., sort.)
permutation.test.gbm(object, n.trees)
gbm.loss(y,f,w,offset,dist,baseline, group, max.rank)
```

Arguments

`object`	a gbm object created from an initial call to [gbm]().
`n.trees`	the number of trees to use for computations. If not provided, the the function will guess: if a test set was used in fitting, the number of trees resulting in lowest test set error will be used; otherwise, if cross-validation was performed, the number of trees resulting in lowest cross-validation error will be used; otherwise, all trees will be used.
`scale.`	whether or not the result should be scaled. Defaults to `FALSE`.
`sort.`	whether or not the results should be (reverse) sorted. Defaults to `FALSE`.
`y,f,w,offset,dist,baseline`	
	For gbm.loss: These components are the outcome, predicted value, observation weight, offset, distribution, and comparison loss function, respectively.
`group, max.rank`	
	Used internally when `distribution = \'pairwise\'`.

Details

This is not intended for end-user use. These functions offer the different methods for computing the relative influence in [summary.gbm](). gbm.loss is a helper function for permutation.test.gbm.

Value

By default, returns an unprocessed vector of estimated relative influences. If the `scale.` and `sort.` arguments are used, returns a processed version of the same.

Author(s)

Greg Ridgeway <gregridgeway@gmail.com>

References

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

L. Breiman (2001). [Random Forests]().

See Also

[summary.gbm]()

---

shrink.gbm   *L1 shrinkage of the predictor variables in a GBM*

Description

Performs recursive shrinkage in each of the trees in a GBM fit using different shrinkage parameters for each variable.

Usage

```
shrink.gbm(object,
           n.trees,
           lambda = rep(10, length(object$var.names)),
           ...)
```

Arguments

object	A gbm.object
n.trees	the number of trees to use
lambda	a vector with length equal to the number of variables containing the shrinkage parameter for each variable
...	other parameters (ignored)

Details

This function is currently experimental. Used in conjunction with a gradient ascent search for inclusion of variables.

Value

predF	Predicted values from the shrunken tree
objective	The value of the loss function associated with the predicted values
gradient	A vector with length equal to the number of variables containing the derivative of the objective function with respect to beta, the logit transform of the shrinkage parameter for each variable

Warning

This function is experimental.

Author(s)

Greg Ridgeway <gregridgeway@gmail.com>

References

Hastie, T. J., and Pregibon, D. Shrinking Trees. AT&T Bell Laboratories Technical Report (March 1990).

See Also

shrink.gbm.pred, gbm

---

shrink.gbm.pred   *Predictions from a shrunked GBM*

Description

Makes predictions from a shrunken GBM model.

Usage

```
shrink.gbm.pred(object,
                newdata,
                n.trees,
                lambda = rep(1, length(object$var.names)),
                ...)
```

Arguments

object	a gbm.object
newdata	dataset for predictions
n.trees	the number of trees to use
lambda	a vector with length equal to the number of variables containing the shrinkage parameter for each variable
...	other parameters (ignored)

Value

A vector with length equal to the number of observations in newdata containing the predictions

Warning

This function is experimental

Author(s)

Greg Ridgeway <gregridgeway@gmail.com>

See Also

shrink.gbm, gbm

---

summary.gbm   *Summary of a gbm object*

Description

Computes the relative influence of each variable in the gbm object.

Usage

```
## S3 method for class 'gbm'
summary(object,
        cBars=length(object$var.names),
        n.trees=object$n.trees,
        plotit=TRUE,
        order=TRUE,
        method=relative.influence,
        normalize=TRUE,
        ...)
```

Arguments

object	a gbm object created from an initial call to gbm.
cBars	the number of bars to plot. If order=TRUE the only the variables with the cBars largest relative influence will appear in the barplot. If order=FALSE then the first cBars variables will appear in the plot. In either case, the function will return the relative influence of all of the variables.
n.trees	the number of trees used to generate the plot. Only the first n.trees trees will be used.
plotit	an indicator as to whether the plot is generated.
order	an indicator as to whether the plotted and/or returned relative influences are sorted.
method	The function used to compute the relative influence. relative.influence is the default and is the same as that described in Friedman (2001). The other current (and experimental) choice is permutation.test.gbm. This method randomly permutes each predictor variable at a time and computes the associated reduction in predictive performance. This is similar to the variable importance measures Breiman uses for random forests, but gbm currently computes using the entire training dataset (not the out-of-bag observations).
normalize	if FALSE then summary.gbm returns the unnormalized influence.
...	other arguments passed to the plot function.

Details

For `distribution="gaussian"` this returns exactly the reduction of squared error attributable to each variable. For other loss functions this returns the reduction attributeable to each varaible in sum of squared error in predicting the gradient on each iteration. It describes the relative influence of each variable in reducing the loss function. See the references below for exact details on the computation.

Value

Returns a data frame where the first component is the variable name and the second is the computed relative influence, normalized to sum to 100.

Author(s)

Greg Ridgeway <gregridgeway@gmail.com>

References

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

L. Breiman (2001).Random Forests.

See Also

gbm

---

validate.gbm   *Test the* gbm *package.*

Description

Run tests on gbm functions to perform logical checks and reproducibility.

Usage

```
validate.gbm()
```

Details

The function uses functionality in the `RUnit` package. A fairly small validation suite is executed that checks to see that relative influence identifies sensible variables from simulated data, and that predictions from GBMs with Gaussian, Cox or binomial distributions are sensible,

Value

An object of class `RUnitTestData`. See the help for `RUnit` for details.

Note

The test suite is not comprehensive.

Author(s)

Harry Southworth

See Also

[gbm](gbm)

Examples

```
# Uncomment the following lines to run - commented out to make CRAN happy
#library(RUnit)
#val <- validate.texmex()
#printHTMLProtocol(val, "texmexReport.html")
```

Index