Package: gbm3 (via r-universe)

October 13, 2026

Version 3.0.2

Date 2026-07-13

Title Generalized Boosted Regression Models

Depends R (>= 3.5.0)

Imports survival, lattice, splines, Rcpp (>= 1.0.0)

Suggests testthat (>= 3.1.0), knitr, rmarkdown, MASS

VignetteBuilder knitr

Description Extensions to Freund and Schapire's AdaBoost algorithm, Y. Freund and R. Schapire (1997) <doi:10.1006/jcss.1997.1504> and Friedman's gradient boosting machine, J.H. Friedman (2001) <doi:10.1214/aos/1013203451>. Includes regression methods for least squares, absolute loss, t-distribution loss, quantile regression, logistic, Poisson, Cox proportional hazards partial likelihood, AdaBoost exponential loss, Huberized hinge loss, and Learning to Rank measures (LambdaMART).

License GPL (>= 2)

URL https://gbm-developers.github.io/gbm3/, https://github.com/gbm-developers/gbm3

Encoding UTF-8

LinkingTo Rcpp

RoxygenNote 8.0.0

NeedsCompilation yes

Author James Hickey [aut], Paul Metcalfe [aut], Greg Ridgeway [aut, cre], Stefan Schroedl [aut], Harry Southworth [aut], Terry Therneau [aut]

Maintainer Greg Ridgeway <gridge@upenn.edu>

Repository https://cran.r-universe.dev

Date/Publication 2026-07-13 19:00:02 UTC

RemoteUrl https://github.com/cran/gbm3

RemoteRef HEAD

RemoteSha 7b75740bb9f5b3228e2f914e8c7feeb5b8e1497e

Contents

available_distributions . . . 2
baseline_hazard . . . 3
calibrate_plot . . . 4
create_dist . . . 6
distribution_name . . . 6
gbm.fit . . . 7
gbm.perf . . . 14
gbm_dist . . . 16
gbm_more . . . 17
gbm_object . . . 18
gbm_roc_area . . . 19
gbmParallel . . . 20
gbmt . . . 21
gbmt_fit . . . 24
gbmt_performance . . . 26
interact . . . 26
iteration_error . . . 28
perf_pairwise . . . 28
permutation_relative_influence . . . 29
plot.GBMFit . . . 30
plot.GBMTPerformance . . . 31
predict.GBMCVFit . . . 32
predict.GBMFit . . . 33
pretty_gbm_tree . . . 34
print.GBMFit . . . 35
quantile_rug . . . 36
relative_influence . . . 37
summary.GBMFit . . . 38
to_old_gbm . . . 39
training_params . . . 40
trees . . . 41

Index **42**

`available_distributions` *Available Distributions*

Description

Returns a vector of the distributions implemented in the package.

Usage

```
available_distributions()
```

Value

a vector containing the names of the available distributions in the package.

Author(s)

James Hickey

`baseline_hazard` *Baseline hazard function*

Description

Computes the Breslow estimator of the baseline hazard function for a proportional hazard regression model - only for censored survival data.

Usage

```
baseline_hazard(surv_times, delta, coxph_preds, eval_times=NULL, smooth=FALSE,
cumulative=TRUE)
```

Arguments

`surv_times`	the survival times - an atomic vector of doubles
`delta`	the censoring indicator - a vector same length as surv_times
`coxph_preds`	the predicted values of the regression model on the log hazard scale
`eval_times`	values at which the baseline hazard will be evaluated
`smooth`	if `TRUE` `baseline_hazard` 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)). `gbmt` can estimate the f(x) component via partial likelihood. After estimating f(x), `baseline_hazard` can compute a nonparametric estimate of lambda(t).

Value

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

Author(s)

James Hickey, Greg Ridgeway <gregridgeway@gmail.com>

References

N. Breslow (1972). "Discussion 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, gbmt

`calibrate_plot` *Calibration plot*

Description

An experimental diagnostic tool that plots the fitted values versus the actual average values. If `distribution` is "Bernoulli" or "Poisson", then the predictions are converted to the response scale (probability or rate). For all other distributions, the calibration plot uses least squares and predicts an expected value.

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 the 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

Value

calibrate.plot returns no values.

Author(s)

James Hickey, 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

```
dataSim <- data.frame(x=rnorm(1000))
dataSim$y <- with(dataSim, rbinom(1000, 1, 1/(1+exp(-(x-0.5*x^2)))))

# showing poor calibration of a linear model
glm1 <- glm(y~x, data=dataSim, family=binomial)
p <- predict(glm1, type="response")
calibrate_plot(dataSim$y, p, xlim=c(0,1), ylim=c(0,1))

# showing better calibration with quadratic
glm1 <- glm(y~poly(x,2), data=dataSim, family=binomial)
p <- predict(glm1,type="response")
calibrate_plot(dataSim$y, p, xlim=c(0,1), ylim=c(0,1))
```

create_dist *Create distributions*

Description

Creation methods that take an empty distribution object and builds on top of the skeleton.

Usage

```
create_dist(empty_obj, ...)
```

Arguments

empty_obj	An empty distribution object of the correct class.
...	extra parameters used to define the distribution object, see gbm_dist.

Value

an appropriated gbm distribution object

Author(s)

James Hickey

distribution_name *What is the distribution name used here?*

Description

What is the distribution name used here?

Usage

```
distribution_name(obj, ...)
```

Arguments

obj	the object for which the distribution name is needed
...	other parameters

Value

a string identifying the distribution

gbm.fit *Generalized Boosted Regression Modeling*

Description

Fits generalized boosted regression models.

Usage

```
gbm.fit(
  x,
  y,
  offset = 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,
  mFeatures = NULL,
  keep.data = TRUE,
  verbose = TRUE,
  var.names = NULL,
  response.name = "y",
  group = NULL,
  tied.times.method = "efron",
  prior.node.coeff.var = 1000,
  strata = NA,
  obs.id = 1:nrow(x)
)

gbm(
  formula = formula(data),
  distribution = "bernoulli",
  data = list(),
  weights,
  subset = NULL,
  offset = NULL,
  var.monotone = NULL,
  n.trees = 100,
  interaction.depth = 1,
  n.minobsinnode = 10,
  shrinkage = 0.001,
```

```
  bag.fraction = 0.5,
  train.fraction = 1,
  mFeatures = NULL,
  cv.folds = 0,
  keep.data = TRUE,
  verbose = FALSE,
  class.stratify.cv = NULL,
  n.cores = NULL,
  par.details = getOption("gbm.parallel"),
  fold.id = NULL,
  tied.times.method = "efron",
  prior.node.coeff.var = 1000,
  strata = NA,
  obs.id = 1:nrow(data)
)
```

Arguments

x, y	For gbm.fit: x is a data frame or data matrix containing the predictor variables and y is a matrix of outcomes. Excluding coxph this matrix of outcomes collapses to a vector; in the case of coxph, it is a Surv(time, status) or Surv(start, stop, status) object. The number of rows in x must be the same as the length of the 1st dimension of y.
offset	an optional model offset
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 parameters needed. If not specified, gbm will try to guess: if the response has only two unique values, bernoulli is assumed (two-factor responses are converted to 0,1); otherwise, if the response has class "Surv", coxph is assumed; otherwise, gaussian is assumed. Available distributions are "gaussian" (squared error), "laplace" (absolute loss), "tdist" (t-distribution loss), "bernoulli" (logistic regression for 0-1 outcomes), "huberized" (Huberized hinge loss for 0-1 outcomes), "adaboost" (the AdaBoost exponential loss for 0-1 outcomes), "poisson" (count outcomes), "coxph" (right-censored or counting-process survival 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. Non-constant weights are unsupported. If "tdist" is specified, the default degrees of freedom is four and this can be controlled 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 in-

stances from the same group and with different target labels can be considered. `metric` is the IR measure to use, one of

list("conc") Fraction of concordant pairs; for binary labels, this is equivalent to the Area under the ROC Curve

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

list("mrr") Mean reciprocal rank of the highest-ranked positive instance

: Mean reciprocal rank of the highest-ranked positive instance

list("map") Mean average precision, a generalization of `mrr` to multiple positive instances

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

list("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).

`w` For `gbm.fit`: `w` is a vector of weights of the same length as the 1st dimension of `y`.

`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.

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

`n.minobsinnode` minimum number of observations (not total weights) in the terminal nodes of the trees.

`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 independent training observations (or patients) randomly selected to propose the next tree in the expansion, depending on the obs.id vector multiple training data rows may belong to a single 'patient'. This introduces randomness 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 ensures the same model can be reconstructed. Preferably, the user can save the returned GBMFit object using save.

nTrain — An integer representing the number of unique patients, each patient could have multiple rows associated with them, 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.

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.

mFeatures — Each node will be trained on a random subset of mFeatures number of features. Each node will consider a new random subset of features, adding variability to tree growth and reducing computation time. mFeatures will be bounded between 1 and nCols. Values outside of this bound will be set to the lower or upper limits.

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.

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.

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'.

tied.times.method — For gbm and gbm.fit: This is an optional string used with CoxPH distribution specifying what method to employ when dealing with tied times. Currently only "efron" and "breslow" are available; the default value is "efron". Setting the string to any other value reverts the method to the original CoxPH model implementation where ties are not explicitly dealt with.

prior.node.coeff.var — Optional double only used with the coxph distribution. It is a prior on the coefficient of variation associated with the hazard rate assigned to each terminal node when fitting a tree.Increasing its value emphasises the importance of the training data in the node when assigning a prediction to said node.

strata — Optional vector of integers (or factors) only used with the coxph distributions. Each integer in this vector represents the stratum the corresponding row in the data belongs to, e. g. if the 10th element is 3 then the 10th data row belongs to the 3rd strata.

obs.id — Optional vector of integers used to specify which rows of data belong to individual patients. Data is then bagged by patient id; the default sets each row of the data to belong to an individual patient.

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.

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. The weights 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.
subset	an optional vector defining a subset of the data to be used
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 and calculate an estimate of generalization error returned in cv_error.
class.stratify.cv	whether the cross-validation should be stratified by class. Is only implemented for bernoulli. The purpose of stratifying the cross-validation is to help avoiding situations in which training sets do not contain all classes.
n.cores	number of cores to use for parallelization. Please use par.details instead; this argument is only provided for convenience.
par.details	Details of the parallelization to use in the core algorithm.
fold.id	An optional vector of values identifying what fold each observation is in. If supplied, cv.folds can be missing. Note: Multiple observations to the same patient must have the same fold id.

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 GBMFit object, and a variety of other loss functions that had not previously had associated boosting algorithms, including the Cox partial likelihood for right-censored or counting-process survival data, the poisson likelihood for count outcomes, and a gradient boosting implementation to minimize the AdaBoost exponential loss function.

gbm is a deprecated function that now acts as a front-end to gbmt_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 over gbm; however gbmt and gbmt_fit are now the current APIs.

Value

gbm and gbm.fit return a GBMFit object.

Functions

- `gbm.fit()`: Core fitting code, for experts only.

Author(s)

James Hickey, Greg Ridgeway <gregridgeway@gmail.com>

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

t-distribution 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.

See Also

`gbmt`, `gbmt_fit` `gbmt_performance`, `plot`, `predict.GBMFit`, `summary.GBMFit`, `pretty_gbm_tree`, `gbmParallel`.

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
    mFeatures = 3,               # half of the features are considered at each node
    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
# , par.details=gbmParallel(num_threads=15) # option for gbm3 to parallelize
    )

# check performance using an out-of-bag estimator
# OOB underestimates the optimal number of iterations
best_iter <- gbmt_performance(gbm1,method="OOB")
print(best_iter)

# check performance using a 50% heldout test set
best_iter <- gbmt_performance(gbm1,method="test")
print(best_iter)

# check performance using 3-fold cross-validation
best_iter <- gbmt_performance(gbm1,method="cv")
print(best_iter)

# plot the performance # plot variable influence
summary(gbm1, num_trees=1)         # based on the first tree
summary(gbm1, num_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$params$num_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
oldpar <- par(no.readonly = TRUE)
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)
par(oldpar) # reset graphics options to previous settings

# do another 100 iterations
gbm2 <- gbm_more(gbm1,100,
                 is_verbose=FALSE) # stop printing detailed progress
```

---

gbm.perf *GBMT Performance*

Description

Estimates optimal number of boosting iterations given a GBMFit object and optionally plots various performance measures.

Usage

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

Arguments

object	a GBMFit object created from an initial call to gbmt or 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 the train error (in black) and the validation 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.
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.
main	the main title for the plot. Defaults to main = "".

Value

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

See Also

gbmt gbmt_performance plot.GBMTPerformance

gbm_dist *GBM Distribution*

Description

Generates distribution object for gbmt.

Usage

```
gbm_dist(name="Gaussian", ...)
```

Arguments

name	The name (a string) of the distribution to be initialized and used in fitting a gradient boosted model via gbmt. The current distributions available can be viewed using the function available_distributions. If no distribution is specified this function constructs a Gaussian distribution by default.
...	Extra named parameters required for initializing certain distributions. If t-distribution is selected, an additional parameter (df) specifying the number of degrees of freedom can be given. The default degrees of freedom is set to four. If quantile is selected then the quantile to estimate may be specified using the named parameter alpha. The default quantile to estimate is 0.25. If the Tweedie distribution is selected the power-law specifying the distribution may be set via the named parameter: power. This parameter defaults to 1.5. The values 0, 1, and 2 are not permitted (use the Gaussian, Poisson, or Gamma distribution instead) If a Cox proportional hazards model is selected, the response must be a right-censored Surv(time, status) object or a counting-process Surv(start, stop, status) object. A number of additional parameters are available: strata A vector of integers (or factors) specifying which strata each data-row belongs to, if none is specified it is assumed all training data is in the same stratum. ties String specifying the method to be used when dealing with tied event times. Currently only "breslow" and "efron" are available, with the latter being the default. prior_node_coeff_var It is a prior on the coefficient of variation associated with the hazard rate assigned to each terminal node when fitting a tree. Increasing its value emphasizes the importance of the training data in the node when assigning a prediction to said node. This defaults to 1000. Finally, if the pairwise distribution is selected a number of parameters also need to be specified. These parameters are group, metric and max_rank. The first 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). For training, only pairs of instances from the same group and with different target labels may be considered. metric is the IR measure to use, one of

list("conc") Fraction of concordant pairs; for binary labels, this is equivalent to the Area under the ROC Curve

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

list("mrr") Mean reciprocal rank of the highest-ranked positive instance

: Mean reciprocal rank of the highest-ranked positive instance

list("map") Mean average precision, a generalization of `mrr` to multiple positive instances

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

list("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.

Value

returns a `GBMDist` object.

Author(s)

James Hickey

`gbm_more` *Perform additional boosting*

Description

Method to perform additional boosting using a `GBMFit` object - does not support further cross validation.

Usage

```
gbm_more(
  gbm_fit_obj,
  num_new_trees = 100,
  data = NULL,
  weights = NULL,
  offset = NULL,
  is_verbose = FALSE
)
```

Arguments

gbm_fit_obj	a GBMFit object produced using gbmt. This object describes the boosted model on which to perform additional boosting.
num_new_trees	a positive integer specifying how many additional iterations to perform. This has a default value of 100.
data	a data.frame or matrix containing the new values for the predictor and response variables for the additional iterations. The names of the variables must match those appearing in the original fit (as well as the number of rows), and this value defaults to NULL. With a value of NULL the original data may be used for the additional boosting, if no original or new data is specified an error will be thrown.
weights	an atomic vector of doubles specifying the importance of each row of the data in the additional iterations. If the previous data used is kept within gbm_fit_obj; then the weights are extracted from the stored GBMData object.
offset	an atomic vector of doubles specifying the offset for each response value in the data used for additional boosting.
is_verbose	a logical specifying whether or not the additional fitting should run "noisely" with feedback on progress provided to the user.

Value

the input GBMFit object with additional iterations provided for the fit.

---

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 gbmt_performance
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 GBMFit object.

Author(s)

Greg Ridgeway <gregridgeway@gmail.com>

See Also

gbmt

---

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)
```

Arguments

obs	Observed value
pred	Predicted value

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

Examples

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

---

gbmParallel *Control parallelization options*

Description

gbmt uses openmp to parallelize its core algorithm, and the details are controlled by this object. As guidance, set num_threads to the number of cores on your computer, and set array_chunk_size to a reasonable - not necessarily small - size.

Usage

```
gbmParallel(num_threads = 1, array_chunk_size = 1024)
```

Arguments

num_threads	the number of threads to use (a positive integer). The number of cores on your computer is a reasonable default.
array_chunk_size	the size of chunks to use in array scans. Values that are too small result in a great deal of overhead; The default of 1024 appears reasonable, but do experiment.

Value

an object of type gbmParallel

---

gbmt *GBMT*

Description

Fits generalized boosted regression models - new API. This prepares the inputs, performing tasks such as creating cv folds, before calling gbmt_fit to call the underlying C++ and fit a generalized boosting model.

Usage

```
gbmt(
  formula,
  distribution = gbm_dist("Gaussian"),
  data,
  weights = rep(1, nrow(data)),
  offset = rep(0, nrow(data)),
  train_params = training_params(num_trees = 2000, interaction_depth = 3,
    min_num_obs_in_node = 10, shrinkage = 0.001, bag_fraction = 0.5, id =
   seq_len(nrow(data)), num_train = round(0.5 * nrow(data)), num_features = ncol(data) -
    1),
  var_monotone = NULL,
  var_names = NULL,
  cv_folds = 1,
  cv_class_stratify = FALSE,
  fold_id = NULL,
  keep_gbm_data = FALSE,
  par_details = getOption("gbm.parallel"),
  is_verbose = FALSE
)
```

Arguments

formula	a symbolic description of the model to be fit. The formula may include an offset term (e.g. y~offset(n) + x).

distribution	a GBMDist object specifying the distribution and any additional parameters needed. If not specified then the distribution will be guessed.
data	a data frame containing the variables in the model. By default, the variables are taken from the environment.
weights	optional vector of weights used in the fitting process. These weights must be positive but need not be normalized. By default they are set to 1 for each data row.
offset	optional vector specifying the model offset; must be positive. This defaults to a vector of 0's, the length of which is equal to the number rows of data.
train_params	a GBMTrainParams object which specifies the parameters used in growing decision trees.
var_monotone	optional vector, the same length as the number of predictors, indicating the relationship each variable has with the outcome. It have a monotone increasing (+1) or decreasing (-1) or an arbitrary relationship.
var_names	a vector of strings of containing the names of the predictor variables.
cv_folds	a positive integer specifying the number of folds to be used in cross-validation of the gbm fit. If cv_folds > 1 then cross-validation is performed; the default of cv_folds is 1.
cv_class_stratify	a bool specifying whether or not to stratify via response outcome. Currently only applies to "Bernoulli" distribution and defaults to false.
fold_id	An optional vector of values identifying what fold each observation is in. If supplied, cv_folds can be missing. Note: Multiple rows of the same observation must have the same fold_id.
keep_gbm_data	a bool specifying whether or not the gbm_data object created in this method should be stored in the results.
par_details	Details of the parallelization to use in the core algorithm (gbmParallel).
is_verbose	if TRUE, gbmt will print out progress and performance of the fit.

Value

a GBMFit object.

Examples

```
## create some data
N <- 1000
X1 <- runif(N)
X2 <- runif(N)
X3 <- factor(sample(letters[1:4],N,replace=TRUE))
mu <- c(-1,0,1,2)[as.numeric(X3)]

p <- 1/(1+exp(-(sin(3*X1) - 4*X2 + mu)))
Y <- rbinom(N,1,p)

# random weights if you want to experiment with them
```

```
w <- rexp(N)
w <- N*w/sum(w)

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


# takes longer, but num_trees=3000 preferable
train_params <-
     training_params(num_trees = 3000,
                     shrinkage = 0.001,
                     bag_fraction = 0.5,
                     num_train = N/2,
                     id=seq_len(nrow(data)),
                     min_num_obs_in_node = 10,
                     interaction_depth = 3,
                     num_features = 3)


# for the example to run quickly, num_trees=100
train_params <-
     training_params(num_trees = 100,
                     shrinkage = 0.001,
                     bag_fraction = 0.5,
                     num_train = N/2,
                     id=seq_len(nrow(data)),
                     min_num_obs_in_node = 10,
                     interaction_depth = 3,
                     num_features = 3)

# fit initial model
gbm1 <- gbmt(Y~X1+X2+X3,               # formula
             data=data,                # dataset
             weights=w,
             var_monotone=c(0,0,0),    # -1: monotone decrease,
                                       # +1: monotone increase,
                                       #  0: no monotone restrictions
             distribution=gbm_dist("Bernoulli"),
             train_params = train_params,
             cv_folds=5,               # do 5-fold cross-validation
             is_verbose = FALSE)          # don't print progress

# plot the performance
#   returns out-of-bag estimated best number of trees
best.iter.oob <- gbmt_performance(gbm1,method="OOB")
plot(best.iter.oob)
print(best.iter.oob)

# returns 5-fold cv estimate of best number of trees
best.iter.cv <- gbmt_performance(gbm1,method="cv")
plot(best.iter.cv)
print(best.iter.cv)

# returns test set estimate of best number of trees
```

```
best.iter.test <- gbmt_performance(gbm1,method="test")
plot(best.iter.cv)
print(best.iter.test)

best.iter <- best.iter.test

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

# create marginal plots
# plot variable X1,X2,X3 after "best" iterations
oldpar <- par(no.readonly = TRUE)
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))
plot(gbm1,1:2,best.iter) # contour plot vars 1 & 2 after "best" num iterations
plot(gbm1,2:3,best.iter) # lattice plot vars 2 & 3 after "best" num iterations

# 3-way plot
plot(gbm1,1:3,best.iter)

# print the first and last trees
print(pretty_gbm_tree(gbm1,1))
print(pretty_gbm_tree(gbm1, gbm1$params$num_trees))
par(oldpar) # reset graphics options to previous settings
```

---

gbmt_fit *GBMT fit*

Description

Fits a generalized boosting model. This is for "power" users who have a large number of variables who wish to avoid calling `model.frame` which can be slow in this instance.

Usage

```
gbmt_fit(
  x,
  y,
  distribution = gbm_dist("Gaussian"),
  weights = rep(1, nrow(x)),
  offset = rep(0, nrow(x)),
  train_params = training_params(num_trees = 100, interaction_depth = 3,
    min_num_obs_in_node = 10, shrinkage = 0.001, bag_fraction = 0.5, id =
    seq_len(nrow(x)), num_train = round(0.5 * nrow(x)), num_features = ncol(x)),
  response_name = "y",
```

```
  var_monotone = NULL,
  var_names = NULL,
  keep_gbm_data = FALSE,
  cv_folds = 1,
  cv_class_stratify = FALSE,
  fold_id = NULL,
  par_details = getOption("gbm.parallel"),
  is_verbose = FALSE
)
```

Arguments

x	a data frame or data matrix containing the predictor variables.
y	is a matrix of outcomes. Excluding CoxPH this matrix of outcomes collapses to a vector; in the case of CoxPH, it is a `Surv(time, status)` or `Surv(start, stop, status)` object. The length of the 1st dimension of y must match the number of rows in x.
distribution	a `GBMDist` object specifying the distribution and any additional parameters needed.
weights	optional vector of weights used in the fitting process. These weights must be positive but need not be normalized. By default they are set to 1 for each data row.
offset	optional vector specifying the model offset; must be positive. This defaults to a vector of 0's, the length of which is equal to the rows of x.
train_params	a GBMTrainParams object which specifies the parameters used in growing decision trees.
response_name	a string specifying the name of the response - defaults to "y".
var_monotone	optional vector, the same length as the number of predictors, indicating the relationship each variable has with the outcome. It have a monotone increasing (+1) or decreasing (-1) or an arbitrary relationship.
var_names	a vector of strings of containing the names of the predictor variables.
keep_gbm_data	a bool specifying whether or not the gbm_data object created in this method should be stored in the results.
cv_folds	a positive integer specifying the number of folds to be used in cross-validation of the gbm fit. If cv_folds > 1 then cross-validation is performed; the default of cv_folds is 1.
cv_class_stratify	a bool specifying whether or not to stratify via response outcome. Currently only applies to "Bernoulli" distribution and defaults to false.
fold_id	An optional vector of values identifying what fold each observation is in. If supplied, cv_folds can be missing. Note: Multiple rows of the same observation must have the same fold_id.
par_details	Details of the parallelization to use in the core algorithm.
is_verbose	if TRUE, gbmt will print out progress and performance of the fit.

Value

a GBMFit object.

---

gbmt_performance *Get performance details for gbm fit*

Description

gbmt_performance estimates the optimal number of boosting iterations from a model fit by gbmt. The precise method used depends on the method parameter.

Usage

```
gbmt_performance(gbm_fit_obj, method)
```

Arguments

gbm_fit_obj	a GBMFit created from an initial call to gbmt.
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 gbmt was called with cv_folds>1.

Value

a GBMTPerformance object, which is a number - the optimal iteration number - with various attributes.

---

interact *Estimate the strength of interaction effects*

Description

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

Usage

```
interact(gbm_fit_obj, data, var_indices=1, num_trees=gbm_fit_obj$params$num_trees)

## S3 method for class 'GBMFit'
interact(
  gbm_fit_obj,
  data,
  var_indices = 1,
  num_trees = gbm_fit_obj$params$num_trees
)
```

Arguments

gbm_fit_obj	a GBMFit object fitted using a call to gbmt.
data	the dataset used to construct gbm_fit_obj. If the original dataset is large, a random subsample may be used to accelerate the computation.
var_indices	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 gbmt formula.
num_trees	the number of trees used to generate the plot. Only the first num_trees trees will be used.

Details

interact.GBMFit 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

gbmt

iteration_error *Extract errors from GBMFit objects*

Description

Extract errors from GBMFit objects

Usage

```
iteration_error(obj, which = c("train", "valid", "cv"))
```

Arguments

obj	a GBMFit object
which	which error measure to extract

Value

a vector giving the error by tree

perf_pairwise *Performance for pairwise*

Description

Additional performance using appropriate metric for pairwise distribution fit.

Usage

```
perf_pairwise(y, f, group, metric = "ndcg", w = NULL, max_rank = 0)
```

Arguments

y	responses used for fit.
f	the predicted responses.
group	the indices specifiying which response variables belong to which groups.
metric	What type of performance measure to compute in perf_pairwise. Can take values "ir_measure_conc", "ir_measure_mrr", "ir_measure_map" or "ir_measure_ndgc".
w	is the weights of each observation.
max_rank	the maximum rank considered in the groups measure.

Value

returns a numeric value of the selected performance metric

Author(s)

Greg Ridgeway <gregridgeway@gmail.com>

See Also

gbmt

`permutation_relative_influence` *Relative influence via permutation*

Description

This function offers a method for computing the relative influence in `summary.GBMFit`, and is not intended to be called directly.

Usage

```
permutation_relative_influence(
  gbm_fit_obj,
  num_trees,
  rescale = FALSE,
  sort_it = FALSE
)
```

Arguments

`gbm_fit_obj`	a `GBMFit` object from an initial call to `gbmt`.
`num_trees`	the number of trees to use for computations. If not provided, 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.
`rescale`	whether or not the result should be scaled. Defaults to `FALSE`.
`sort_it`	whether or not the results should be (reverse) sorted. Defaults to `FALSE`.

Details

Calculates the relative influence of predictors via random permutation of each predictor one at a time and calculating the associated reduction in predictive performance. This experimental measure is similar to the variable importance measures Breiman uses for random forests, but `gbmt` currently computes using the entire training dataset (not the out-of-bag observations).

Value

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

Author(s)

Greg Ridgeway <gregridgeway@gmail.com>

See Also

summary.GBMFit Random Forests.

---

plot.GBMFit *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 'GBMFit'
plot(
  x,
  var_index = 1,
  num_trees = gbm_fit_obj$params$num_trees,
  continuous_resolution = 100,
  grid_levels = NULL,
  return_grid = FALSE,
  type = "link",
  ...
)
```

Arguments

x	a GBMFit object fitted using a call to gbmt
var_index	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 gbmt formula. If length(var_index) is between 1 and 3 then plot.GBMFit produces the plots. Otherwise, plot.GBMFit returns only the grid of evaluation points and their average predictions
num_trees	the number of trees used to generate the plot. Only the first num_trees trees will be used
continuous_resolution	The number of equally space points at which to evaluate continuous predictors
grid_levels	A list containing the points at which to evaluate each predictor in var_index (in the same order as var_index). For continuous predictors this is usually a regular sequence of values within the range of the variable. For categorical predictors, the points are the levels of the factor. When length(var_index) is one, the values can be provided directly, outside a list. This is NULL by default and generated automatically from the data, using continuous_resolution for

	continuous predictors. Forcing the values can be useful to evaluate two models on the same exact range
return_grid	if TRUE then plot.GBMFit 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_gmt
...	other arguments passed to the plot function

Details

plot.GBMFit produces low dimensional projections of the GBMFit object, see gbmt, by integrating out the variables not included in the var_index 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_gbmt 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.GBMFit produces no graphics and only returns the grid of evaluation points and their average predictions.

References

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

See Also

gbmt, plot

---

plot.GBMTPerformance *Plot GBM performance details*

Description

The train and validation error (in black and red respectively) are plotted against the iteration number. If the initial model was built with cross-validation, the cross-validation error is shown in green.

Usage

```
## S3 method for class 'GBMTPerformance'
plot(x, out_of_bag_curve = FALSE, overlay = TRUE, main = "", ...)
```

Arguments

x	a GBMTPerformance object (created by gbmt_performance)
out_of_bag_curve	indicates whether to plot the out-of-bag performance measures in a second plot.
overlay	if TRUE and out_of_bag_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.
main	the main title for the plot.
...	currently ignored

Details

The scale of the error measurement, shown on the left vertical axis, depends on the distribution argument used in the initial call to gbmt.

Value

No return value, only plot generation

---

predict.GBMCVFit *Predictions for CV fitted GBM models*

Description

Calculate predictions from cross validated generalized boosting model from gbm2.

Usage

```
## S3 method for class 'GBMCVFit'
predict(object, gbm_data_obj, cv_folds, cv_group, best_iter_cv, ...)
```

Arguments

object	a GBMCVFit object containing CV gbm models
gbm_data_obj	a GBMData object containing all of the data used to fit a gbm model.
cv_folds	a positive integer specifying the number of folds to be used in cross-validation of the gbm fit.
cv_group	vector of integers specifying which row of data belongs to which cv_fold.
best_iter_cv	number of trees with the smallest cv error
...	not currently used

Value

a matrix of predictions for each cv fold.

Author(s)

James Hickey

`predict.GBMFit` *Predict method for GBM Model Fits*

Description

Predicted values based on a generalized boosted model object - from gbmt

Usage

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

Arguments

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

Details

`predict.GBMFit` produces predicted values for each observation in a new dataset `newdata` using 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 `gbmt` do not include the offset term. The user may add the value of the offset to the predicted value if desired.

If `gbm_fit_obj` was fit using `gbmt`, 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 `gbmt` 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.

See Also

gbmt

---

pretty_gbm_tree *Print gbm tree components*

Description

GBMFit 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(gbm_fit_obj, tree_index = 1)
```

Arguments

gbm_fit_obj	a GBMFit object initially fit using gbmt.
tree_index	the index of the tree component to extract from gbm_fit_obj 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 gbm_fit_obj$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)

James Hickey, Greg Ridgeway <gregridgeway@gmail.com>

See Also

gbm, gbmt

print.GBMFit *Print model summary*

Description

Display basic information about a GBMFit object.

Usage

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

Arguments

x	is a fitted generalized boosting object of class GBMFit.
...	arguments passed to print.default.

Details

Prints some information about the model object. In particular, this method prints the call to gbmt, 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 bernoulli or adaboost was used, the confusion matrix and prediction accuracy are printed (objects being allocated to the class with highest probability for bernoulli). These classifications are performed using the cross-validation fitted values.

If the 'distribution' was specified as gaussian, laplace, quantile or t-distribution, a summary of the residuals is displayed. The residuals are the cross-validation residuals. Also, a pseudo R-squared value is displayed. For Gaussian response, this is 1 - sum(r*r) / sum(z*z) where z = y - mean(y). For the other distributions, this is 1 - (median(abs(r)) / mad(y))^2, following the suggestion of Rousseeuw and Leroy (equation 3.11). Note that this definition of a robust R-squared is contentious.

Value

Returns x invisibly

Author(s)

James Hickey, Harry Southworth, Daniel Edwards

References

P. J. Rousseeuw and A. M. Leroy, Robust Regression and Outlier Detection, Wiley, 1987 (2003).

See Also

gbmt

---

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.

Examples

```
x <- rnorm(100)
y <- rnorm(100)
plot(x,y)
quantile_rug(x)
```

`relative_influence` *Methods for estimating relative influence*

Description

This function offers a method for computing the relative influence in `summary.GBMFit`, and is not intended to be called directly.

Usage

```
relative_influence(gbm_fit_obj, num_trees, rescale = FALSE, sort_it = FALSE)
```

Arguments

`gbm_fit_obj`	a `GBMFit` object created from an initial call to `gbmt`.
`num_trees`	the number of trees to use for computations. If not provided, 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.
`rescale`	whether or not the result should be scaled. Defaults to `FALSE`.
`sort_it`	whether or not the results should be (reverse) sorted. Defaults to `FALSE`.

Details

`relative_influence` is the same as that described in Friedman (2001). `permutation_relative_influence` 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 `gbmt` currently computes using the entire training dataset (not the out-of-bag observations).

Value

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

Author(s)

James Hickey, 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.GBMFit

---

summary.GBMFit *Summary of a GBMFit object*

Description

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

Usage

```
## S3 method for class 'GBMFit'
summary(
  object,
  cBars = length(object$variables$var_names),
  num_trees = length(trees(object)),
  plot_it = TRUE,
  order_it = TRUE,
  method = relative_influence,
  normalize = TRUE,
  ...
)
```

Arguments

object	a GBMFit object created from an initial call to gbmt.
cBars	the number of bars to plot. If order_it=TRUE then only the cBars variables with the largest relative influence will appear in the barplot. If order_it=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.
num_trees	the number of trees used to generate the plot. Only the first num_trees trees will be used.
plot_it	an indicator as to whether the plot is generated.
order_it	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_relative_influence. 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 GBMGaussianDist this returns exactly the reduction of squared error attributable to each variable. For other loss functions this returns the reduction attributable to each variable 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)

James Hickey, 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

gbmt

---

to_old_gbm *Convert GBMFit to previous gbm object*

Description

Function that takes a GBMFit object produced from a call to gbmt and converts it to the form of a gbm object from Version 2.1 of the package.

Usage

```
to_old_gbm(gbm_fit_obj)
```

Arguments

gbm_fit_obj	a GBMFit object produced by a call to gbmt.

Value

a gbm object of the form from Version 2.1 of the package.

Author(s)

James Hickey

`training_params` *Training parameters*

Description

Class that contains the training parameters for the gbm model

Usage

```
training_params(
  num_trees = 100,
  interaction_depth = 1,
  min_num_obs_in_node = 10,
  shrinkage = 0.001,
  bag_fraction = 0.5,
  num_train = (2 * min_num_obs_in_node + 1)/bag_fraction + 1,
  id = seq_len(num_train),
  num_features = 1
)
```

Arguments

`num_trees`	Number of trees used in the fit.
`interaction_depth`	Maximum depth of each tree
`min_num_obs_in_node`	Minimum number of observations each node in a tree must have.
`shrinkage`	shrinkage parameter applied to each tree in the expansion. Also known as the learning rate or step-size reduction.
`bag_fraction`	fraction of independent training observations selected to create the next tree in the expansion. Introduces randomness in the model fit; if bag_fraction < 1 then running the same model twice will result in similar but different fits.
`num_train`	number of obs of data used in training the model. This defaults to the minimum number of observations allowed - `(2*min_num_obs_in_node + 1)/bag_fraction + 1`.
`id`	optional vector of integers, specifying which rows in the data correspond to which observations. Individual observations may have many rows of data associated with them. This defaults to `seq_len(num_train)`. NB: When calling `gbmt` or `gbmt_fit` the id should be the default.
`num_features`	number of random features/columns to use in training model. This defaults to `1`.

Value

training parameters object

Author(s)

James Hickey

`trees` *Extract trees from GBMFit objects*

Description

Extract trees from GBMFit objects

Usage

```
trees(obj)
```

Arguments

`obj`	a GBMFit object

Value

a vector containing the trees generated in the fit

Index

∗ **aplot**
quantile_rug, 36
∗ **hplot**
calibrate_plot, 4
permutation_relative_influence, 29
plot.GBMFit, 30
relative_influence, 37
summary.GBMFit, 38
∗ **methods**
baseline_hazard, 3
gbm_object, 18
interact, 26
∗ **models**
gbm.fit, 7
gbm_roc_area, 19
predict.GBMFit, 33
print.GBMFit, 35
∗ **nonlinear**
gbm.fit, 7
gbm.perf, 14
perf_pairwise, 28
print.GBMFit, 35
∗ **nonparametric**
gbm.fit, 7
gbm.perf, 14
perf_pairwise, 28
print.GBMFit, 35
∗ **print**
pretty_gbm_tree, 34
∗ **regression**
predict.GBMFit, 33
∗ **survival**
baseline_hazard, 3
gbm.fit, 7
gbm.perf, 14
perf_pairwise, 28
print.GBMFit, 35
∗ **tree**
gbm.fit, 7
gbm.perf, 14
perf_pairwise, 28

available_distributions, 2

baseline_hazard, 3

calibrate_plot, 4
create_dist, 6

distribution_name, 6

gbm, *15*, *20*, *34*
gbm *(*gbm.fit*)*, 7
gbm.fit, 7
gbm.perf, 14
gbm_conc *(*gbm_roc_area*)*, 19
gbm_dist, *6*, 16
gbm_more, *10*, *11*, 17
gbm_object, 18
gbm_roc_area, 19
gbmParallel, *12*, 20, *22*
gbmt, *3*, *4*, *12*, *15*, *18*, *19*, 21, *26*, *27*, *29–34*, *36–40*
gbmt_fit, *12*, 24, *40*
gbmt_performance, *12*, *15*, *19*, 26, *32*

interact, 26
iteration_error, 28

jitter, *36*

lattice, *31*

model.frame, *11*

ns, *5*

perf_pairwise, 28
permutation_relative_influence, 29, *37*, *38*
plot, *12*, *31*, *36*

plot.GBMFit, 30
plot.GBMTPerformance, *15*, 31
polygon, *5*
predict.GBMCVFit, 32
predict.GBMFit, *12*, 33
pretty_gbm_tree, *12*, *19*, 34
print *(*print.GBMFit*)*, 35
print.GBMFit, 35

quantile, *36*
quantile_rug, 36

relative_influence, 37, *37, 38*
rug, *5*, *36*

save, *10*
summary.GBMFit, *12*, *29, 30*, *37*, 38, *38*
supsmu, *3*
survfit, *4*

to_old_gbm, 39
training_params, 40
trees, 41