Package 'Rborist'

February 3, 2025

Title Extensible, Parallelizable Implementation of the Random Forest Algorithm

Version 0.3-11

Date 2025-2-2

Maintainer Mark Seligman <mseligman@suiji.org>

BugReports https://github.com/suiji/Arborist/issues

Description Scalable implementation of classification and regression forests, as described by Breiman (2001), <DOI:10.1023/A:1010933404324>.

URL https://github.com/suiji/Rborist.CRAN, https://github.com/suiji/Arborist

License MPL (>= 2) | GPL (>= 2) | file LICENSE

LazyLoad yes

Depends R(>= 3.3)

Imports Rcpp (>= 0.12.2), data.table (>= 1.9.8), digest

Suggests testthat, knitr, rmarkdown, markdown

VignetteBuilder knitr

LinkingTo Rcpp

NeedsCompilation yes

Author Mark Seligman [aut, cre]

Repository CRAN

Date/Publication 2025-02-02 23:10:16 UTC

Contents

`expandfe` *Expands forest values into front-end readable vectors.*

Description

Formats training output into a form suitable for illustration of feature contributions.

Usage

```
 ## Default S3 method:
expandfe(arbOut)
```

Arguments

`arbOut`	an object of type `rfTrain` produced by training.

Value

An object of type `ExpandReg` or `ExpandCtg` containing human-readable representations of the trained forest.

Author(s)

Mark Seligman at Suiji.

Examples

```
  ## Not run:
    data(iris)
    rb <- Rborist(iris[,-5], iris[,5])
    ffe <- expandfe(rb)

    # An rfTrain counterpart is NYI.

## End(Not run)
```

Export *Exportation Format for rfArb Training Output*

Description

Formats training output into a form suitable for illustration of feature contributions.

Usage

```
 ## Default S3 method:
Export(arbOut)
```

Arguments

arbOut	an object of type Rborist produced by training.

Value

An object of type Export.

Author(s)

Mark Seligman at Suiji.

Examples

```
  ## Not run:
    data(iris)
    rb <- Rborist(iris[,-5], iris[,5])
    ffe <- Export(rb)

## End(Not run)
```

forestWeight *Meinshausen forest weights*

Description

Normalized observation counts across a prediction set.

Usage

```
## Default S3 method:
forestWeight(objTrain, prediction, sampler=objTrain$sampler,
nThread=0, verbose = FALSE, ...)
```

Arguments

objTrain	an object of class rfArb, created from a previous invocation of the command Rborist or rfArb to train.
prediction	an object of class SummaryReg or SummaryCtg obtained from prediction using objTrain and argument indexing=TRUE.
sampler	an object of class Sampler, as documented for command of the same name.
nThread	specifies a prefered thread count.
verbose	whether to output progress of weighting.
...	not currently used.

Value

a numeric matrix having rows equal to the Meinshausen weight of each new datum.

Author(s)

Mark Seligman at Suiji.

References

Meinshausen, N. (2016) Quantile Random Forests. Journal of Machine Learning Research 17(1), 1-68.

See Also

Rborist

Examples

```
## Not run:
  # Regression example:
  nRow <- 5000
  x <- data.frame(replicate(6, rnorm(nRow)))
  y <- with(x, X1^2 + sin(X2) + X3 * X4) # courtesy of S. Welling.
  rb <- Rborist(x,y)


  newdata <- data.frame(replace(6, rnorm(nRow)))

  # Performs separate prediction on new data, saving indices:
  pred <- predict(rb, newdata, indexing=TRUE)
  weights <- forestWeight(rb, pred)

  obsIdx <- 215 # Arbitrary observation index (zero-based row number)

  # Inner product should equal prediction, modulo numerical vagaries:
  yPredApprox <- weights[obsIdx,] %*% y
  print((yPredApprox - pred$yPred[obsIdx])/yPredApprox)
```

```
## End(Not run)
```

`predict.arbTrain` *predict method for arbTrain result*

Description

Prediction and test using Rborist.

Usage

```
## S3 method for class 'arbTrain'
predict(object, newdata, sampler, yTest=NULL,
keyedFrame = FALSE, quantVec=numeric(0), quantiles = length(quantVec) > 0,
ctgCensus = "votes", indexing = FALSE, trapUnobserved = FALSE,
bagging = FALSE, nThread = 0, verbose = FALSE, ...)
```

Arguments

`object`	an object of class `arbTrain`, created from a previous invocation of the command `rfArb`, `Rborist` or `rfTrain` to train.
`newdata`	a design frame or matrix containing new data, with the same signature of predictors as in the training command.
`sampler`	an object of class `Sampler` used in the command.
`yTest`	a response vector against which to test the new predictions.
`keyedFrame`	whether the columns of `newdata` may appear in arbitrary order or as a superset of the predictors used to train.
`quantVec`	a vector of quantiles to predict.
`quantiles`	whether to predict quantiles.
`ctgCensus`	whether/how to summarize per-category predictions. "votes" specifies the number of trees predicting a given class. "prob" specifies a normalized, probabilistic summary. "probSample" specifies sample-weighted probabilities, similar to quantile histogramming.
`indexing`	whether to record the final node index, typically terminal, of tree traversal.
`trapUnobserved`	reports score for nonterminal upon encountering values not observed during training, such as missing data.
`bagging`	whether prediction is restricted to out-of-bag samples.
`nThread`	suggests ans OpenMP-style thread count. Zero denotes default processor setting.
`verbose`	whether to output progress of prediction.
`...`	not currently used.

Value

an object of one of two classes:

- `SummaryReg` summarizing regression, consisting of:
  - `prediction` an object of class `PredictReg` consisting of:
    - `yPred` the estimated numerical response.
    - `qPred` quantiles of prediction, if requested.
    - `qEst` quantile of the estimate, if quantiles requested.
    - `indices` final index of prediction, if requested.
  - `validation` if validation requested, an object of class `ValidReg` consisting of:
    - `mse` the mean-squared error of the estimate.
    - `rsq` the r-squared statistic of the estimate.
    - `mae` the mean absolute error of the estimate.
  - `importance` if permution importance requested, an object of class `importanceReg`, containing multiple instances of:
    - `names` the predictor names.
    - `mse` the per-predictor mean-squared error, under permutation.
- `SummaryCtg` summarizing classification, consisting of:
  - `PredictCtg` consisting of:
    - `yPred` estimated categorical response.
    - `census` factor-valued matrix of the estimate, by category, if requested.
    - `prob` matrix of estimate probabilities, by category, if requested.
    - `indices` final index of prediction, if requested.
  - `validation` if validation requested, an object of class `ValidCtg` consisting of:
    - `confusion` the confusion matrix.
    - `misprediction` the misprediction rate.
    - `oobError` the out-of-bag error.
  - `importance` if permution importance requested, an object of class `importanceCtg`, consisting of:
    - `mispred` the misprediction rate, by predictor.
    - `oobErr` the out-of-bag error, by predictor.

Author(s)

Mark Seligman at Suiji.

See Also

`rfTrain`

Examples

```
## Not run:
  # Regression example:
  nRow <- 5000
  x <- data.frame(replicate(6, rnorm(nRow)))
  y <- with(x, X1^2 + sin(X2) + X3 * X4) # courtesy of S. Welling.

  pf <- preformat(x)
  sp <- presample(y)
  rb <- arbTrain(pf, sp, y)


  # Performs separate prediction on new data:
  xx <- data.frame(replace(6, rnorm(nRow)))
  pred <- predict(rb, xx)
  yPred <- pred$yPred

  rb <- Rborist(x,y)

  # Performs separate prediction on new data:
  xx <- data.frame(replacate(6, rnorm(nRow)))
  pred <- predict(rb, xx)
  yPred <- pred$yPred

  # As above, but also records final indices of each tree walk:
  #
  pred <- predict(rb, xx, indexing=TRUE)
  print(pred$indices[c(1:2), ])


  # As above, but predicts over \code{newdata} with unobserved values.
  # In the case of numerical data, only missing values are considered
  # unobserved.  Missing values are encoded as \code{NaN}, which are
  # incomparable, precipitating \code{false} on every test.  Prediction
  # therefore takes the \code{false} branch when encountering missing
  # values:
  #
  xxMissing <- xx
  xxMissing[6, c(15, 32, 87, 101)] <- NA
  pred <- predict(rb, xxMissing)


  # As above, but returns a nonterminal score upon encountering
  # unobserved values. Neither the true nor the false branch from the
  # testing node is taken.  Instead, the score returned is derived
  # from all leaf nodes (terminals) reached by the testing
  # (nonterminal) node.
  #
  pred <- predict(rb, xxMissing, trapUnobserved = TRUE)


  # Performs separate prediction, using original response as test
```

```
# vector:
pred <- predict(rb, xx, y)
mse <- pred$mse
rsq <- pred$rsq


# Performs separate prediction with (default) quantiles:
pred <- predict(rb, xx, quantiles="TRUE")
qPred <- pred$qPred


# Performs separate prediction with deciles:
pred <- predict(rb, xx, quantVec = seq(0.1, 1.0, by = 0.10))
qPred <- pred$qPred


# Classification examples:
data(iris)
rb <- Rborist(iris[-5], iris[5])


# Generic prediction using training set.
# Census as (default) votes:
pred <- predict(rb, iris[-5])
yPred <- pred$yPred
census <- pred$census

# Using the \code{keyedFrame} option allows the columns of
# \code{newdata} to appear in arbitrary order, so long as the
# columns present during training appear as a subset:
#
pred <- predict(rb, iris[c(2, 4, 3, 1)], keyedFrame=TRUE)


# As above, but validation census to report class probabilities:
pred <- predict(rb, iris[-5], ctgCensus="prob")
prob <- pred$prob


# As above, but with training reponse as test vector:
pred <- predict(rb, iris[-5], iris[5], ctgCensus = "prob")
prob <- pred$prob
conf <- pred$confusion
misPred <- pred$misPred

# As above, but predicts nonterminal when encountering categories
# not observed during training.  That is, prediction returns a score
# derived from all terminal nodes (leaves) reached from the
# (nonterminal) testing node.
#
# In this case, "unobserved" refers to categories not present in
# the subpartition over which a splitting is performed.  As training
# partitions the data into smaller and smaller regions, a given
```

```
  # category becomes less likely to appear in a region.
  #
  # More generally, unobserved data can include missing predictors as
  # well as categories appearing in \code{newdata} which were not
  # present during training.
  #
  pred <- predict(rb, trapUnobserved=TRUE)

## End(Not run)
```

`predict.rfArb` *predict method for rfArb result*

Description

Prediction and test using Rborist.

Usage

```
## S3 method for class 'rfArb'
predict(object, newdata, sampler, yTest=NULL,
keyedFrame = FALSE, quantVec=numeric(0), quantiles = length(quantVec) > 0,
ctgCensus = "votes", indexing = FALSE, trapUnobserved = FALSE,
bagging = FALSE, nThread = 0, verbose = FALSE, ...)
```

Arguments

`object`	an object of class `rfArb`, created from a previous invocation of the command `rfArb` or `Rborist` to train.
`newdata`	a design frame or matrix containing new data, with the same signature of predictors as in the training command.
`sampler`	an object of class `Sampler` used in the command.
`yTest`	a response vector against which to test the new predictions.
`keyedFrame`	whether the columns of `newdata` may appear in arbitrary order or as a superset of the predictors used to train.
`quantVec`	a vector of quantiles to predict.
`quantiles`	whether to predict quantiles.
`ctgCensus`	whether/how to summarize per-category predictions. "votes" specifies the number of trees predicting a given class. "prob" specifies a normalized, probabilistic summary. "probSample" specifies sample-weighted probabilities, similar to quantile histogramming.
`indexing`	whether to record the final node index, typically terminal, of tree traversal.
`trapUnobserved`	reports score for nonterminal upon encountering values not observed during training, such as missing data.
`bagging`	whether prediction is restricted to out-of-bag samples.

nThread	suggests ans OpenMP-style thread count. Zero denotes default processor setting.
verbose	whether to output progress of prediction.
...	not currently used.

Value

an object of one of two classes:

- SummaryReg summarizing regression, consisting of:
  - prediction an object of class PredictReg consisting of:
    - yPred the estimated numerical response.
    - qPred quantiles of prediction, if requested.
    - qEst quantile of the estimate, if quantiles requested.
    - indices final index of prediction, if requested.
  - validation if validation requested, an object of class ValidReg consisting of:
    - mse the mean-squared error of the estimate.
    - rsq the r-squared statistic of the estimate.
    - mae the mean absolute error of the estimate.
  - importance if permution importance requested, an object of class importanceReg, containing multiple instances of:
    - names the predictor names.
    - mse the per-predictor mean-squared error, under permutation.
- SummaryCtg summarizing classification, consisting of:
  - PredictCtg consisting of:
    - yPred estimated categorical response.
    - census factor-valued matrix of the estimate, by category, if requested.
    - prob matrix of estimate probabilities, by category, if requested.
    - indices final index of prediction, if requested.
  - validation if validation requested, an object of class ValidCtg consisting of:
    - confusion the confusion matrix.
    - misprediction the misprediction rate.
    - oobError the out-of-bag error.
  - importance if permution importance requested, an object of class importanceCtg, consisting of:
    - mispred the misprediction rate, by predictor.
    - oobErr the out-of-bag error, by predictor.

Author(s)

Mark Seligman at Suiji.

See Also

rfTrain

Examples

```
## Not run:
  # Regression example:
  nRow <- 5000
  x <- data.frame(replicate(6, rnorm(nRow)))
  y <- with(x, X1^2 + sin(X2) + X3 * X4) # courtesy of S. Welling.

  pf <- preformat(x)
  sp <- presample(y)
  rb <- rfArb(pf, sp, y)


  # Performs separate prediction on new data:
  xx <- data.frame(replace(6, rnorm(nRow)))
  pred <- predict(rb, xx)
  yPred <- pred$yPred

  rb <- Rborist(x,y)

  # Performs separate prediction on new data:
  xx <- data.frame(replacate(6, rnorm(nRow)))
  pred <- predict(rb, xx)
  yPred <- pred$yPred

  # As above, but also records final indices of each tree walk:
  #
  pred <- predict(rb, xx, indexing=TRUE)
  print(pred$indices[c(1:2), ])


  # As above, but predicts over \code{newdata} with unobserved values.
  # In the case of numerical data, only missing values are considered
  # unobserved.  Missing values are encoded as \code{NaN}, which are
  # incomparable, precipitating \code{false} on every test.  Prediction
  # therefore takes the \code{false} branch when encountering missing
  # values:
  #
  xxMissing <- xx
  xxMissing[6, c(15, 32, 87, 101)] <- NA
  pred <- predict(rb, xxMissing)


  # As above, but returns a nonterminal score upon encountering
  # unobserved values. Neither the true nor the false branch from the
  # testing node is taken.  Instead, the score returned is derived
  # from all leaf nodes (terminals) reached by the testing
  # (nonterminal) node.
  #
  pred <- predict(rb, xxMissing, trapUnobserved = TRUE)


  # Performs separate prediction, using original response as test
```

```
# vector:
pred <- predict(rb, xx, y)
mse <- pred$mse
rsq <- pred$rsq


# Performs separate prediction with (default) quantiles:
pred <- predict(rb, xx, quantiles="TRUE")
qPred <- pred$qPred


# Performs separate prediction with deciles:
pred <- predict(rb, xx, quantVec = seq(0.1, 1.0, by = 0.10))
qPred <- pred$qPred


# Classification examples:
data(iris)
rb <- Rborist(iris[-5], iris[5])


# Generic prediction using training set.
# Census as (default) votes:
pred <- predict(rb, iris[-5])
yPred <- pred$yPred
census <- pred$census

# Using the \code{keyedFrame} option allows the columns of
# \code{newdata} to appear in arbitrary order, so long as the
# columns present during training appear as a subset:
#
pred <- predict(rb, iris[c(2, 4, 3, 1)], keyedFrame=TRUE)


# As above, but validation census to report class probabilities:
pred <- predict(rb, iris[-5], ctgCensus="prob")
prob <- pred$prob


# As above, but with training reponse as test vector:
pred <- predict(rb, iris[-5], iris[5], ctgCensus = "prob")
prob <- pred$prob
conf <- pred$confusion
misPred <- pred$misPred

# As above, but predicts nonterminal when encountering categories
# not observed during training.  That is, prediction returns a score
# derived from all terminal nodes (leaves) reached from the
# (nonterminal) testing node.
#
# In this case, "unobserved" refers to categories not present in
# the subpartition over which a splitting is performed.  As training
# partitions the data into smaller and smaller regions, a given
```

```
  # category becomes less likely to appear in a region.
  #
  # More generally, unobserved data can include missing predictors as
  # well as categories appearing in \code{newdata} which were not
  # present during training.
  #
  pred <- predict(rb, trapUnobserved=TRUE)

## End(Not run)
```

`preformat` *Preformatting for Training with Warm Starts*

Description

Presorts and formats training frame into a form suitable for subsequent training by `rfArb` caller or `rfTrain` command. Wraps this form to spare unnecessary recomputation when iteratively retraining, for example, under parameter sweep.

Usage

```
## Default S3 method:
preformat(x,
   nThread = 0,
                   verbose=FALSE,
                   ...)
```

Arguments

`x`	the design frame expressed as either a `data.frame` object with numeric and/or `factor` columns or as a numeric or factor-valued matrix.
`nThread`	number of cores to run in parallel, if available.
`verbose`	indicates whether to output progress of preformatting.
`...`	unused.

Value

an object of class `Deframe` consisting of:

- `rleFrame` run-length encoded representation of class `RLEFrame` consisting of:
  - `rankedFrame` run-length encoded representation of class `RankedFrame` consisting of:
    - `nRow` the number of observations encoded.
    - `runVal` the run-length encoded values.
    - `runRow` the corresponding row indices.
    - `rleHeight` the number of encodings, per predictor.
    - `topIdx` the accumulated end index, per predictor.

  - numRanked packed representation of sorted numerical values of class NumRanked consisting of:
    * numVal distinct numerical values.
    * numHeight value offset per predictor.
  - facRanked packed representation of sorted factor values of class FacRanked consisting of:
    * facVal distinct factor values, zero-based.
    * facHeight value offset per predictor.
- nRow the number of training observations.
- signature an object of type Signature consisting of:
  - predForm predictor class names.
  - level per-predictor levels, regardless whether realized.
  - factor per-predictor realized levels.
  - colNames predictor names.
  - rowNames observation names.

Author(s)

Mark Seligman at Suiji.

Examples

```
## Not run:
  data(iris)
  pt <- preformat(iris[,-5])

  ppTry <- seq(0.2, 0.5, by= 0.3/10)
  nIter <- length(ppTry)
  rsq <- numeric(nIter)
  for (i in 1:nIter) {
    rb <- Rborist(pt, iris[,5], predProb=ppTry[i])
    rsq[i] = rb$validiation$rsq
  }

## End(Not run)
```

---

presample *Forest-wide Observation Sampling*

Description

Observations sampled for each tree to be trained. In the case of the Random Forest algorithm, this is the bag.

Usage

```
## Default S3 method:
presample(y,
                              samplingWeight = numeric(0),
                              nSamp = 0,
                              nRep = 500,
                              withRepl =  TRUE,
                              nHoldout = 0,
                              nFold = 1,
                              verbose = FALSE,
                              nTree = 0,
                              ...)
```

Arguments

y	A vector to be sampled, typically the response.
samplingWeight	Per-observation sampling weights. Default is uniform.
nSamp	Size of sample draw. Default draws y length.
nRep	Number of samples to draw. Replaces deprecated nTree.
withRepl	true iff sampling is with replacement.
nHoldout	Number of observations to omit from sampling. Augmented by unobserved response values.
nFold	Number of collections into which to partition the respone.
verbose	true iff tracing execution.
nTree	Number of samples to draw. Deprecated.
...	not currently used.

Value

an object of class Sampler consisting of:

- yTrain the sampled vector.
- nSamp the sample sizes drawn.
- nRep the number of independent samples.
- nTree synonymous with nRep. Deprecated.
- samples a packed data structure encoding the observation index and corresponding sample count.
- hash a hashed digest of the data items.

References

Tille, Yves. Sampling algorithms. Springer New York, 2006.

Examples

```
## Not run:
  y <- runif(1000)

  # Samples with replacement, 500 vectors of length 1000:
  ps <- presample(y)

  # Samples, as above, with 63 observations held out:
  ps <- presample(y, nHoldout = 63)

  # Samples without replacement, 250 vectors of length 500:
  ps2 <- presample(y, nTree=250, nSamp=500, withRepl = FALSE)



## End(Not run)
```

Rborist *Rapid Decision Tree Construction and Evaluation*

Description

Legacy entry for accelerated implementation of the Random Forest (trademarked name) algorithm. Calls the suggested entry, `rfArb`.

Usage

```
## Default S3 method:
 Rborist(x,
              y,
              ...)
```

Arguments

`x`	the design matrix expressed as a `PreFormat` object, as a `data.frame` object with numeric and/or `factor` columns or as a numeric matrix.
`y`	the response (outcome) vector, either numerical or categorical. Row count must conform with `x`.
`...`	specific to `rfArb`.

Value

an object of class `rfArb`, as documented in command of the same name.

Author(s)

Mark Seligman at Suiji.

Examples

```
## Not run:
  # Regression example:
  nRow <- 5000
  x <- data.frame(replicate(6, rnorm(nRow)))
  y <- with(x, X1^2 + sin(X2) + X3 * X4) # courtesy of S. Welling.

  # Classification example:
  data(iris)

  # Generic invocation:
  rb <- Rborist(x, y)

## End(Not run)
```

RboristNews *NEWS Displayer for Rborist*

Description

Displays NEWS associated with Rborist releases.

Usage

```
RboristNews()
```

Value

None.

---

rfArb *Rapid Decision Tree Construction and Evaluation*

Description

Accelerated implementation of the Random Forest (trademarked name) algorithm. Tuned for multicore and GPU hardware. Bindable with most numerical front-end languages in addtion to R. Invocation is similar to that provided by *randomForest* package.

Usage

```
## Default S3 method:
 rfArb(x,
                   y,
                 autoCompress = 0.25,
                 ctgCensus = "votes",
                 classWeight = numeric(0),
                 discardState = FALSE,
                 impPermute = 0,
                 indexing = FALSE,
                 maxLeaf = 0,
                 minInfo = 0.01,
                 minNode = if (is.factor(y)) 2 else 3,
                 nHoldout = 0,
                 nLevel = 0,
                 nSamp = 0,
                 nThread = 0,
                 nTree = 500,
                 noValidate = FALSE,
                 predFixed = 0,
                 predProb = 0.0,
                 predWeight = numeric(0),
                 quantVec = numeric(0),
                 quantiles = length(quantVec) > 0,
                 regMono = numeric(0),
                 rowWeight = numeric(0),
                 samplingWeight = numeric(0),
                 splitQuant = numeric(0),
                 streamline = FALSE,
                 thinLeaves = streamline || (is.factor(y) && !indexing),
                 trapUnobserved = FALSE,
                 treeBlock = 1,
                 verbose = FALSE,
                 withRepl = TRUE,
                 ...)
```

Arguments

x	the design matrix expressed as a PreFormat object, as a data.frame object with numeric and/or factor columns or as a numeric matrix.
y	the response (outcome) vector, either numerical or categorical. Row count must conform with x.
autoCompress	plurality above which to compress predictor values.
ctgCensus	report categorical validation by vote or by probability.
classWeight	proportional weighting of classification categories.
discardState	minimizes storage by discarding primary training output. Useful for parameter sweeps and cross-validation, in which only validation may be of interest.

impPermute	number of importance permutations: 0 or 1.
indexing	whether to report final index, typically terminal, of validation tree traversal.
maxLeaf	maximum number of leaves in a tree. Zero denotes no limit.
minInfo	information ratio with parent below which node does not split.
minNode	minimum number of distinct row references to split a node.
nHoldout	number of observations to omit from sampling. Augmented by missing response values.
nLevel	maximum number of tree levels to train, including terminals (leaves). Zero denotes no limit.
nSamp	number of rows to sample, per tree.
nThread	suggests an OpenMP-style thread count. Zero denotes the default processor setting.
nTree	the number of trees to train.
noValidate	whether to train without validation.
predFixed	number of trial predictors for a split (mtry).
predProb	probability of selecting individual predictor as trial splitter.
predWeight	relative weighting of individual predictors as trial splitters.
quantVec	quantile levels to validate.
quantiles	whether to report quantiles at validation.
regMono	signed probability constraint for monotonic regression.
rowWeight	row weighting for initial sampling of tree. Deprecated
samplingWeight	row weighting for initial sampling of tree.
splitQuant	(sub)quantile at which to place cut point for numerical splits
.	
streamline	whether to streamline sampler contents to save space.
thinLeaves	bypasses creation of leaf state in order to reduce storage footprint.
trapUnobserved	reports score for nonterminal upon encountering values not observed during training, such as missing data.
treeBlock	maximum number of trees to train during a single level (e.g., coprocessor computing).
verbose	indicates whether to output progress of training.
withRepl	whether row sampling is by replacement.
...	not currently used.

Value

an object sharing classes rfArb, a supplementary collection consisting of the following items:

- sampler an object of class Sampler, as described in the documentation for the presample command, that summarizes the bagging structure.

- `training` a list summarizing the training task, consisting of the following fields:
  - `call` the calling invocation.
  - `info` a vector of forest-wide Gini (classification) or weighted variance (regression), by predictor.
  - `version` the version of the `Rborist` package used to train.
  - `diag` diagnostics accumulated over the training task.
  - `samplerHash` hash value of the `Sampler` object used to train. Recorded for consistency of subsequent commands.
- `prediction` an object of class `PredictReg` or `PredictCtg`, as described by the documention for command `predict`.
- `validation` an object of class `ValidReg` or `ValidCtg`, as described by the documention for command`validate`, if validation is requested.
- `importance` an object of class `ImportanceReg` or`ImportanceCtg`, as described by the documention for command `predict`, if permutation performance has been requested.

Author(s)

Mark Seligman at Suiji.

References

Breiman, L. (2001) Random Forests, Machine Learning 45(1), 5-32.

See Also

`Rborist`

Examples

```
## Not run:
  # Regression example:
  nRow <- 5000
  x <- data.frame(replicate(6, rnorm(nRow)))
  y <- with(x, X1^2 + sin(X2) + X3 * X4) # courtesy of S. Welling.

  # Classification example:
  data(iris)

  # Generic invocation:
  rb <- rfArb(x, y)


  # Causes 300 trees to be trained:
  rb <- rfArb(x, y, nTree = 300)


  # Causes rows to be sampled without replacement:
  rb <- rfArb(x, y, withRepl=FALSE)
```

```
# Causes validation census to report class probabilities:
rb <- rfArb(iris[-5], iris[5], ctgCensus="prob")


# Applies table-weighting to classification categories:
rb <- rfArb(iris[-5], iris[5], classWeight = "balance")


# Weights first category twice as heavily as remaining two:
rb <- rfArb(iris[-5], iris[5], classWeight = c(2.0, 1.0, 1.0))


# Does not split nodes when doing so yields less than a 2% gain in
# information over the parent node:
rb <- rfArb(x, y, minInfo=0.02)


# Does not split nodes representing fewer than 10 unique samples:
rb <- rfArb(x, y, minNode=10)


# Trains a maximum of 20 levels:
rb <- rfArb(x, y, nLevel = 20)


# Trains, but does not perform subsequent validation:
rb <- rfArb(x, y, noValidate=TRUE)


# Chooses 500 rows (with replacement) to root each tree.
rb <- rfArb(x, y, nSamp=500)


# Chooses 2 predictors as splitting candidates at each node (or
# fewer, when choices exhausted):
rb <- rfArb(x, y, predFixed = 2)


# Causes each predictor to be selected as a splitting candidate with
# distribution Bernoulli(0.3):
rb <- rfArb(x, y, predProb = 0.3)


# Causes first three predictors to be selected as splitting candidates
# twice as often as the other two:
rb <- rfArb(x, y, predWeight=c(2.0, 2.0, 2.0, 1.0, 1.0))


# Causes (default) quantiles to be computed at validation:
rb <- rfArb(x, y, quantiles=TRUE)
qPred <- rb$validation$qPred
```

```
  # Causes specfied quantiles (deciles) to be computed at validation:
  rb <- rfArb(x, y, quantVec = seq(0.1, 1.0, by = 0.10))
  qPred <- rb$validation$qPred


  # Constrains modelled response to be increasing with respect to X1
  # and decreasing with respect to X5.
  rb <- rfArb(x, y, regMono=c(1.0, 0, 0, 0, -1.0, 0))


  # Causes rows to be sampled with random weighting:
  rb <- rfArb(x, y, samplingWeight=runif(nRow))


  # Suppresses creation of detailed leaf information needed for
  # quantile prediction and external tools.
  rb <- rfArb(x, y, thinLeaves = TRUE)

  # Directs prediction to take a random branch on encountering
  # values not observed during training, such as NA or an
  # unrecognized category.

  predict(rb, trapUnobserved = FALSE)

  # Directs prediction to silently trap unobserved values, reporting a
  # score associated with the current nonterminal tree node.

  predict(rb, trapUnobserved = TRUE)

  # Sets splitting position for predictor 0 to far left and predictor
  # 1 to far right, others to default (median) position.

  spq <- rep(0.5, ncol(x))
  spq[0] <- 0.0
  spq[1] <- 1.0
  rb <- rfArb(x, y, splitQuant = spq)

## End(Not run)
```

---

rfTrain *Rapid Decision Tree Training*

Description

Accelerated training using the Random Forest (trademarked name) algorithm. Tuned for multicore and GPU hardware. Bindable with most numerical front-end languages in addtion to R.

Usage

```
## Default S3 method:
rfTrain(preFormat,
                 sampler,
                 y,
                autoCompress = 0.25,
                ctgCensus = "votes",
                classWeight = numeric(0),
                maxLeaf = 0,
                minInfo = 0.01,
                minNode = if (is.factor(y)) 2 else 3,
                nLevel = 0,
                nThread = 0,
                predFixed = 0,
                predProb = 0.0,
                predWeight = numeric(0),
                regMono = numeric(0),
                splitQuant = numeric(0),
                thinLeaves = FALSE,
                treeBlock = 1,
                verbose = FALSE,
                ...)
```

Arguments

y	the response (outcome) vector, either numerical or categorical.
preFormat	Compressed, presorted representation of the predictor values. Row count must conform with y.
sampler	Compressed representation of the sampled response.
autoCompress	plurality above which to compress predictor values.
ctgCensus	report categorical validation by vote or by probability.
classWeight	proportional weighting of classification categories.
maxLeaf	maximum number of leaves in a tree. Zero denotes no limit.
minInfo	information ratio with parent below which node does not split.
minNode	minimum number of distinct row references to split a node.
nLevel	maximum number of tree levels to train, including terminals (leaves). Zero denotes no limit.
nThread	suggests an OpenMP-style thread count. Zero denotes the default processor setting.
predFixed	number of trial predictors for a split (mtry).
predProb	probability of selecting individual predictor as trial splitter.
predWeight	relative weighting of individual predictors as trial splitters.
regMono	signed probability constraint for monotonic regression.

splitQuant	(sub)quantile at which to place cut point for numerical splits
.	
thinLeaves	bypasses creation of leaf state in order to reduce memory footprint.
treeBlock	maximum number of trees to train during a single level (e.g., coprocessor computing).
verbose	indicates whether to output progress of training.
...	Not currently used.

Value

an object of class arbTrain, containing:

- version the version of the Rborist package used to train.
- samplerHash hash value of the Sampler object used to train. Recorded for consistency of subsequent commands.
- predInfo a vector of forest-wide Gini (classification) or weighted variance (regression), by predictor.
- forest an object of class Forest containing:
  - nTree the number of trees trained.
  - node an object of class Node consisting of:
    - treeNode forest-wide vector of packed node representations.
    - extent per-tree node counts.
    - scores numeric vector of scores, for all terminals and nonterminals.
    - factor an object of class Factor consisting of:
      - facSplit forest-wide vector of packed factor bits.
      - extent per-tree extent of factor bits.
      - observed forest-wide vector of observed factor bits.
  - Leaf an object of class Leaf containing:
    - extent forest-wide vector of leaf populations, i.e., counts of unique samples.
    - index forest-wide vector of sample indices.
- diag diagnostics accumulated over the training task.

Author(s)

Mark Seligman at Suiji.

See Also

Rborist

Examples

```
## Not run:
  # Regression example:
  nRow <- 5000
  x <- data.frame(replicate(6, rnorm(nRow)))
  y <- with(x, X1^2 + sin(X2) + X3 * X4) # courtesy of S. Welling.

  # Classification example:
  data(iris)

  # Generic invocation:
  rt <- rfTrain(y)


  # Causes 300 trees to be trained:
  rt <- rfTrain(y, nTree = 300)


  # Causes validation census to report class probabilities:
  rt <- rfTrain(iris[-5], iris[5], ctgCensus="prob")


  # Applies table-weighting to classification categories:
  rt <- rfTrain(iris[-5], iris[5], classWeight = "balance")


  # Weights first category twice as heavily as remaining two:
  rt <- rfTrain(iris[-5], iris[5], classWeight = c(2.0, 1.0, 1.0))


  # Does not split nodes when doing so yields less than a 2% gain in
  # information over the parent node:
  rt <- rfTrain(y, preFormat, sampler, minInfo=0.02)


  # Does not split nodes representing fewer than 10 unique samples:
  rt <- rfTrain(y, preFormat, sampler, minNode=10)


  # Trains a maximum of 20 levels:
  rt <- rfTrain(y, preFormat, sampler, nLevel = 20)


  # Trains, but does not perform subsequent validation:
  rt <- rfTrain(y, preFormat, sampler, noValidate=TRUE)


  # Chooses 500 rows (with replacement) to root each tree.
  rt <- rfTrain(y, preFormat, sampler, nSamp=500)


  # Chooses 2 predictors as splitting candidates at each node (or
```

```
  # fewer, when choices exhausted):
  rt <- rfTrain(y, preFormat, sampler, predFixed = 2)


  # Causes each predictor to be selected as a splitting candidate with
  # distribution Bernoulli(0.3):
  rt <- rfTrain(y, preFormat, sampler, predProb = 0.3)


  # Causes first three predictors to be selected as splitting candidates
  # twice as often as the other two:
  rt <- rfTrain(y, preFormat, sampler, predWeight=c(2.0, 2.0, 2.0, 1.0, 1.0))


  # Constrains modelled response to be increasing with respect to X1
  # and decreasing with respect to X5.
  rt <- rfTrain(x, y, preFormat, sampler, regMono=c(1.0, 0, 0, 0, -1.0, 0))


  # Suppresses creation of detailed leaf information needed for
  # quantile prediction and external tools.
  rt <- rfTrain(y, preFormat, sampler, thinLeaves = TRUE)

  spq <- rep(0.5, ncol(x))
  spq[0] <- 0.0
  spq[1] <- 1.0
  rt <- rfTrain(y, preFormat, sampler, splitQuant = spq)

## End(Not run)
```

`Streamline.rfArb` *Reducing Memory Footprint of Trained Decision Forest*

Description

Clears fields deemed no longer useful.

Usage

```
## S3 method for class 'rfArb'
Streamline(arbOut)
```

Arguments

`arbOut`	Trained forest object of class `rfArb`.

Value

an object of class `rfArb` with sample data cleared.

Author(s)

Mark Seligman at Suiji.

Examples

```
## Not run:
    ## Trains.
    rs <- Rborist(x, y)
    ...
    ## Replaces trained object with streamlined copy.
    rs <- Streamline(rs)

## End(Not run)
```

---

validate *Separate Validation of Trained Decision Forest*

Description

Permits trained decision forest to be validated separately from training.

Usage

```
## Default S3 method:
validate(train, preFormat, sampler = NULL,  ctgCensus
= "votes", impPermute = 0, quantVec = numeric(0), quantiles =
length(quantVec) > 0, indexing = FALSE, trapUnobserved = FALSE, nThread = 0, verbose =
FALSE, ...)
```

Arguments

`train`	an object of class `Rborist` obtained from previous training.
`sampler`	summarizes the response and its per-tree samplgin.
`preFormat`	internal representation of the design matrix, of class `PreFormat`
`ctgCensus`	report categorical validation by vote or by probability.
`impPermute`	specifies the number of importance permutations: 0 or 1.
`quantVec`	quantile levels to validate.
`quantiles`	whether to report quantiles at validation.
`indexing`	whether to report final index, typically terminal, of tree traversal.
`trapUnobserved`	indicates whether to return a nonterminal for values unobserved during training, such as missing data.
`nThread`	suggests an OpenMP-style thread count. Zero denotes the default processor setting.
`verbose`	indicates whether to output progress of validation.
`...`	not currently used.

Value

either of two pairs of objects:

- SummaryReg summarizing regression, as documented with the command predict.arbTrain.
- validation an object of class ValidReg consisting of:
  - mse the mean-square error of the estimate.
  - rsq the r-squared statistic of the estimate.
  - mae the mean absolute error of the estimate.
- SummaryCtg summarizing classification, as documented with the command predict.arbTrain.
- validation an object of class ValidCtg consisting of:
  - confusion the confusion matrix.
  - misprediction the misprediction rate.
  - oobError the out-of-bag error.

Author(s)

Mark Seligman at Suiji.

Examples

```
## Not run:
    ## Trains without validation.
    rb <- Rborist(x, y, novalidate=TRUE)
    ...
    ## Delayed validation using a preformatted object.
    pf <- preformat(x)
    v <- validate(rb, pf)

## End(Not run)
```

Index