Package ‘bigrf’

February 19, 2015

Version 0.1-11
Date 2014-05-16
Title Big Random Forests: Classification and Regression Forests for Large Data Sets
Maintainer Aloysius Lim <aloysius.lim@gmail.com>
OS_type unix
Depends R (>= 2.14), methods, bigmemory
Imports foreach
Suggests MASS, doParallel
LinkingTo bigmemory, BH
Description This is an implementation of Leo Breiman's and Adele Cutler's Random Forest algorithms for classification and regression, with optimizations for performance and for handling of data sets that are too large to be processed in memory. Forests can be built in parallel at two levels. First, trees can be grown in parallel on a single machine using foreach. Second, multiple forests can be built in parallel on multiple machines, then merged into one. For large data sets, disk-based big.matrix's may be used for storing data and intermediate computations, to prevent excessive virtual memory swapping by the operating system. Currently, only classification forests with a subset of the functionality in Breiman and Cutler's original code are implemented. More functionality and regression trees may be added in the future.
License GPL-3
Copyright 2013-2014 Aloysius Lim
URL https://github.com/aloyius-lim/bigrf
BugReports https://github.com/aloyius-lim/bigrf/issues
Author Aloysius Lim [aut, cre],
Leo Breiman [aut],
Adele Cutler [aut]
NeedsCompilation yes
Repository CRAN
Date/Publication 2014-05-16 07:52:54
Description

This is an implementation of Leo Breiman’s and Adele Cutler’s Random Forest algorithms for classification and regression, with optimizations for performance and for handling of data sets that are too large to be processed in memory. Forests can be built in parallel at two levels. First, trees can be built in parallel on a single machine using `foreach`. Second, multiple forests can be built in parallel on multiple machines, then merged into one. For large data sets, disk-based `big.matrix`'s may be used for storing data and intermediate computations, to prevent excessive virtual memory swapping by the operating system. Currently, only classification forests with a subset of the functionality in Breiman and Cutler’s original code are implemented. More functionality and regression trees will be added in the future. See file INSTALL-WINDOWS in the source package for Windows installation instructions.

Details

Package: bigrf
Version: 0.1-11
Date: 2014-05-16
OS_type: unix
Depends: R (>= 2.14), methods, bigmemory
Imports: foreach
Suggests: MASS, doParallel
LinkingTo: bigmemory, BH  
License: GPL-3  
Copyright: 2013-2014 Aloysius Lim  
URL: https://github.com/aloyisius-lim/bigrf  
BugReports: https://github.com/aloyisius-lim/bigrf/issues

Index:

- **bigforest-class**: Classification Random Forests
- **bigcprediction-class**: Random Forest Predictions
- **bigctree-class**: Classification Trees in Random Forests
- **bigrf-package**: Big Random Forests: Classification and Regression Forests for Large Data Sets
- **bigrfc**: Build a Classification Random Forest Model
- **bigrfprox-class**: Proximity Matrices
- **fastimp-methods**: Compute Fast (Gini) Variable Importance
- **generateSyntheticClass**: Generate Synthetic Second Class for Unsupervised Learning
- **grow-methods**: Grow More Trees in a Random Forest
- **interactions-methods**: Compute Variable Interactions
- **merge-methods**: Merge Two Random Forests
- **outliers-methods**: Compute Outlier Scores
- **predict-methods**: Predict Classes of Test Examples
- **proximities-methods**: Compute Proximity Matrix
- **scaling-methods**: Compute Metric Scaling Co-ordinates
- **varimp-methods**: Compute Variable Importance

The main entry point for this package is **bigrfc**, which is used to build a classification random forest on the given training data and forest-building parameters. **bigrfc** returns the forest as an object of class "bigforest", which contains the trees grown as objects of class "bigctree". After a forest is built, more trees can be grown using **grow**.

**Performance Optimizations**

For better performance, trees may be grown in parallel by registering an appropriate parallel backend for **foreach**. As an example, the following code uses the **doParallel** package to enable tree-growing on all available cores on the machine. This code must be executed before calling **bigrfc** or **grow**. See **foreach** for more details on supported parallel backends.

```r
library(doParallel)
registerDoParallel(cores=detectCores(all.tests=TRUE))
```
Multiple random forests can also be built in parallel on multiple machines (using the same training
data and parameters), then merged into one forest using `merge`.

For large data sets, the training data, intermediate computations and some outputs (e.g. proximity
matrices) may be cached on disk using `big.matrix` objects. This enables random forests to be
built on fairly large data sets without hitting RAM limits, which will cause excessive virtual memory
swapping by the operating system.

Disk caching may be turned off for optimal performance on smaller data sets by setting function /
method argument `cachepath` to `NULL`, causing the `big.matrix`’s to be created in memory.

**Author(s)**

Original Fortran77 code by Leo Breiman and Adele Cutler.
R port with disk caching and parallelization enhancements by Aloysius Lim.
Maintainer: Aloysius Lim

**References**


**See Also**

`randomForest` `cforest`
factorvars: Object of class "logical". Indicates which variables are factors or categorical (TRUE), and which are numeric (FALSE).

columns in big.matrix a. Meant for internal use by bigrf or grow when growing trees.

y: Object of class "factor". Class labels for the training set.

ytable: Object of class "table". Counts of training examples in each class.

ntrees: Object of class "integer". Number of trees in the forest.

nsplitvar: Object of class "integer". Number of variables to split on at each node.

maxndsize: Object of class "integer". Maximum number of examples in each node when growing the trees.

maxeslevels: Object of class "integer". Maximum number of levels for categorical variables for which exhaustive search of possible splits will be performed.

nrandsplit: Object of class "integer". Number of random splits to examine for categorical variables with more than maxeslevels levels.

oobtimes: Object of class "integer". Number of times each training example has been out-of-bag.

oobvotes: Object of class "matrix". Out-of-bag votes for each training example.

oobpred: Object of class "integer". Out-of-bag predictions for each training example.

trainclsrcerr: Object of class "numeric". Training errors of out-of-bag examples, by class.

trainerr: Object of class "numeric". Total training error of out-of-bag examples.

trainconfusion: Object of class "table". Confusion matrix for out-of-bag examples.

varginidec: Object of class "numeric". Decrease in Gini impurity for each variable over all trees.

cachepath: Object of class "character or NULL". Path to folder where data caches used in building the forest were stored, or NULL if data was processed completely in memory.

Extends

Class "list", from data part.
Class "vector", by class "list", distance 2.

Methods

grow signature(forest = "bigcforest"): Grow more trees in the random forest, using the same parameters. See grow for details.

merge signature(x = "bigcforest", y = "bigcforest"): Merge two random forests into one. See merge for details.

predict signature(object = "bigcforest"): Predict the classes of a set of test examples. See predict for details.

varimp signature(forest = "bigcforest"): Compute variable importance based on out-of-bag estimates. See varimp for details.
**bigcprediction-class**

Random Forest Predictions

**Description**

Class containing the outputs of `predict`ing on a test set using a random forest.

**Objects from the Class**

Objects can be created by calls of the form `new("bigcprediction", ...), but most often are generated by `predict`.**

**Slots**

- `.Data`: Object of class "integer". The predicted class for each test example.
- `ntest`: Object of class "integer". The number of test examples used for prediction.
- `testlabelled`: Object of class "logical". Whether the test examples were labelled. If TRUE, then error estimates and the confusion matrix are available.
- `ntrees`: Object of class "integer". Number of trees in the random forest.
- `testytable`: Object of class "table". Counts of test examples in each class, if test examples were labelled. Otherwise, NULL.
- `testvotes`: Object of class "matrix". Weighted class votes for each test example. The prediction for each example is the class that received the highest total vote.
- `testclserr`: Object of class "numeric". Prediction error for each class, if test examples were labelled. Otherwise, NULL.
- `testerr`: Object of class "numeric". Total prediction error for all classes, if test examples were labelled. Otherwise, NULL.
- `testconfusion`: Object of class "table". The confusion matrix for the test set, if test examples were labelled. Otherwise, NULL.

**Functions**

- `fastimp` signature(forest = "bigcforest"): Compute fast (Gini) variable importance. See `fastimp` for details.
- `interactions` signature(forest = "bigcforest"): Compute variable interactions. See `interactions` for details.
- `proximities` signature(forest = "bigcforest"): Compute the proximity matrix. See `proximities` for details.
- `prototypes` signature(forest = "bigcforest", prox = "bigrfprox"): Compute class prototypes. See `prototypes` for details.
- `show` signature(object = "bigcforest"): Print the random forest.
- `summary` signature(object = "bigcforest"): Print summary information on the random forest, including out-of-bag training error estimates and the confusion matrix.
bigctree-class

Extends

Class "integer", from data part.
Class "numeric", by class "integer", distance 2.
Class "vector", by class "integer", distance 2.
Class "data.frameRowLabels", by class "integer", distance 2.

Methods

show signature(object = "bigcprediction"): Print prediction results.
summary signature(object = "bigcprediction"): Print summary information on prediction results, including test error estimates and the confusion matrix if test labels were supplied during prediction.

bigctree-class  Classification Trees in Random Forests

Description

Class representing a tree in a classification random forest.

Objects from the Class

Objects can be created by calls of the form new("bigctree", ...), but most often are generated when building a bigcforest using bigrfc.

Slots

insamp: Object of class "integer". Number of times each training example is in the bag for this tree.
inweight: Object of class "numeric". Sum of class weights for each in-bag example.
nodes: Object of class "integer". Number of nodes in the tree.
treemap: Object of class "matrix". Indices of tree nodes. For a given node n, treemap[n, ] represents the indices of the left and right children. If a node is terminal, then both elements of treemap[n, ] are 0.
nodeclass: Object of class "integer". Majority class of each terminal node.
nodewt: Object of class "numeric". Sum of class weights of the examples in each terminal node.
bestvar: Object of class "integer". Best variable that each node was split on, or 0 for terminal nodes.
bestnumsplit: Object of class "numeric". Split point for nodes split on a numerical variable.
bestcatsplit: Object of class "list". Split point, expressed as integer vectors indicating which factor levels to split on, for nodes split on a categorical variable.
termincount: Object of class "numeric". Number of training examples in each terminal node.
trainprednode: Object of class "integer". Terminal node number for each training example.
trainpredclass: Object of class "integer". Class of terminal node for each training example.
tgini: Object of class "numeric". Total gini.
Methods

No methods defined with class "bigctree" in the signature.

---

bigrfc  

*Build a Classification Random Forest Model*

---

Description

Build a classification random forest model using Leo Breiman and Adele Cutler's algorithm, with enhancements for large data sets. This implementation uses the `bigmemory` package for disk-based caching during growing of trees, and the `foreach` package to parallelize the tree-growing process.

Usage

```r
bigrfc(x, y, ntrees = 50L, varselect = NULL, varnlevels = NULL,
       nsplitvar = round(sqrt(ifelse(is.null(varselect), ncol(x),
                                 length(varselect)))), maxeslevels = 11L,
       nrandsplit = 1023L, maxndsize = 1L, yclasswts = NULL,
       printerrfreq = 10L, printclserr = TRUE, cachepath = tempdir(), trace = 0L)
```

Arguments

- **x**: A `big.matrix`, matrix or `data.frame` of predictor variables. If a matrix or `data.frame` is specified, it will be converted into a `big.matrix` for computation.
- **y**: An integer or factor vector of response variables.
- **ntrees**: The number of trees to be grown in the forest, or 0 to build an empty forest to which trees can be added using `grow`. Default: 50.
- **varselect**: An integer vector specifying which columns in `x` to use. If not specified, all variables will be used.
- **varnlevels**: An integer vector with elements specifying the number of levels in the corresponding variables in use, or 0 for numeric variables. Used only when `x` does not contain levels information (i.e. `x` is a `matrix` or `big.matrix`). If `x` is a `data.frame`, `varnlevels` will be inferred from `x`. If `x` is not a `data.frame` and `varnlevels` is `NULL`, all variables will be treated as numeric. If all columns of `x` are used, `varnlevels` should have as many elements as there are columns of `x`. But if `varselect` is specified, then `varnlevels` and `varselect` should be of the same length.
- **nsplitvar**: The number of variables to split on at each node. Default: If `varselect` is specified, the square root of the number of variables specified; otherwise, the square root of the number of columns of `x`.
- **maxeslevels**: Maximum number of levels for categorical variables for which exhaustive search of possible splits will be performed. Default: 11. This will amount to searching \((2^{(11 - 1)}) - 1 = 1,023\) splits.
nrandsplit  Number of random splits to examine for categorical variables with more than maxeslevels levels. Default: 1,023.

maxndsize  Maximum number of examples in each node when growing the trees. Nodes will be split if they have more than this number of examples. Default: 1.

yclasswts  A numeric vector of class weights, or NULL if all classes should be weighted equally.

printerrfreq  An integer, specifying how often error estimates should be printed to the screen. Default: error estimates will be printed every 10 trees.

printclserr  TRUE for error estimates for individual classes to be printed, in addition to the overall error estimates. Default: TRUE.

cachepath  Path to folder where data caches used in building the forest can be stored. If NULL, then the big.matrix's will be created in memory with no disk caching, which would be suitable for small data sets. If caching is used, some of the cached files can be reused in other methods like varimp, shortening method initialization time. If the user wishes to reuse the cached files in this manner, it is suggested that a folder other than tempdir() is used, as the operating system may automatically delete any cache files in tempdir(). Default: tempdir().

trace  0 for no verbose output. 1 to print verbose output on growing of trees. 2 to print more verbose output on processing of individual nodes. Default: 0. Due to the way %dopar% handles the output of the tree-growing iterations, you may not see the verbose output in some GUIs like RStudio. For best results, run R from the command line in order to see all the verbose output.

Value

An object of class "bigcforest" containing the specified number of trees, which are objects of class "bigctree".

References


See Also

randomForest cforest

Examples

# Classify cars in the Cars93 data set by type (Compact, Large,
# Midsize, Small, Sporty, or Van).

# Load data.
data(Cars93, package="MASS")
x <- Cars93
y <- Cars93$Type
# Select variables with which to train model.
vars <- c(4:22)

# Run model, grow 30 trees.
forest <- bigrfc(x, y, ntree=30L, varselect=vars, cachepath=NULL)

---

**bigrfprox-class**  
Proximity Matrices

**Description**

Class representing a proximity matrix for a random forest. Element \([m, n]\) in the matrix indicate the number of times examples \(m\) and \(n\) end up in the same node, averaged over the number of trees in the forest. This is useful for computing scaling co-ordinates, prototypes, and outliers.

**Objects from the Class**

Objects can be created by calls of the form `new("bigrfprox", ...)`, but most often are generated by `proximities`.

"bigrfprox" inherits from "big.matrix". Given a "bigrfprox" object `prox`, each row represents an example, and each column represents the `ncol(prox)` "nearest" examples. If `ncol(prox)` is less than the number of examples, then the examples referred to in each row could be different. In that case, the big.matrix `prox@examples` indicates which examples are referred to in `prox`, i.e. `prox[i, j]` is the proximity between example \(i\) and the example `prox@examples[i, j]`.

**Slots**

- **examples**: Object of class "big.matrix". If the number of columns in the proximity matrix is less than the number of rows, this indicates which examples are referred to in the proximity matrix. Otherwise (i.e. all pairwise proximities were computed), it is NULL.
- **cachepath**: Object of class "character". The cache path for the big.matrix's for the proximity matrix and examples, or NULL if the big.matrix's were created in memory.
- **address**: Object of class "externalptr". Inherited from class "big.matrix".  

**Extends**

Class "big.matrix", directly.

**Methods**

- **scaling** signature(`prox = "bigrfprox"`): Compute metric scaling co-ordinates for training examples in the random forest. See `scaling` for details.
- **prototypes** signature(`forest = "bigcforest", prox = "bigrfprox"`): Compute class prototypes. See `prototypes` for details.
**Description**

Calculates variable importance using a fast method, by adding up the decreases in Gini impurity for each variable over all trees. The results are often consistent with the full variable importance calculated with `varimp`.

**Usage**

```r
## S4 method for signature 'bigcforest'
fastimp(forest)
```

**Arguments**

- `forest`: A random forest of class "bigcforest".

**Value**

A numeric vector containing the fast (Gini) variable importance measures for each variable.

**Methods**

signature(forest = "bigcforest") Compute the fast (Gini) variable importance for a classification random forest.

**References**


**See Also**

`varimp`

**Examples**

```r
# Classify cars in the Cars93 data set by type (Compact, Large, # Midsize, Small, Sporty, or Van).

# Load data.
data(Cars93, package="MASS")
x <- Cars93
y <- Cars93$Type

# Select variables with which to train model.
```
generateSyntheticClass

Generate Synthetic Second Class for Unsupervised Learning

Description

To use Random Forests for unsupervised learning, the training set \( x \) is treated as a single class. This function creates a synthetic second class for classification by sampling at random from the univariate distributions of the original data. This is useful, for example, for clustering.

Usage

```r
generateSyntheticClass(x, ...)
```

Arguments

- **x**
  - A `big.matrix`, `matrix` or `data.frame` containing the predictor variables of the original training set.
- **...**
  - If \( x \) is a `big.matrix`, these arguments will be passed on to `big.matrix` to control how the `big.matrix` for the two-class training set is created.

Value

A list containing the following components:

- **x**
  - The two-class training set, comprising the original training set and the synthesized second class. It will be an object of the same type as the argument \( x \).
- **y**
  - A factor vector that labels the two classes in \( x \).

References


Examples

# Perform unsupervised learning on the Cars93 data set.
# Load data.
data(Cars93, package="MASS")

# Create second synthetic class for unsupervised learning.
newdata <- generateSyntheticClass(Cars93)

# Select variables with which to train model.
vars <- c(4:22)

# Run model, grow 30 trees.
forest <- bigrfc(newdata$x, newdata$y, ntree=30L, varselect=vars,
                 cachepath=NULL)

---

**grow-methods**

*Grow More Trees in a Random Forest*

**Description**

Grow more trees in a random forest, using the same parameters.

**Usage**

```r
## S4 method for signature 'bigforest'
grow(forest, x=NULL, ntrees=50L, printerrfreq=10L,
     printclserr=TRUE, reuse.cache=FALSE, trace=0L)
```

**Arguments**

- `forest`: A random forest of class "bigforest".
- `x`: A `big.matrix`, matrix or data.frame of predictor variables. The data must not have changed, otherwise unexpected modelling results may occur. If a matrix or data.frame is specified, it will be converted into a big.matrix for computation. Optional if `reuse.cache` is TRUE.
- `ntrees`: The number of additional trees to grow in the forest. Default: 50.
- `printerrfreq`: An integer, specifying how often error estimates should be printed to the screen. Default: error estimates will be printed every 10 trees.
- `printclserr`: TRUE for error estimates for individual classes to be printed, in addition to the overall error estimates. Default: TRUE.
- `reuse.cache`: TRUE to reuse disk caches of the big.matrix's `x` and `asave` from the initial building of the random forest, which may significantly reduce initialization time for large data sets. If TRUE, the user must ensure that the files 'x' and 'x.desc' in `forest@cachepath` have not been modified or deleted. Files 'asave' and 'asave.desc' must also be present if there are any numeric variables.
interactions-methods

trace

0 for no verbose output. 1 to print verbose output on growing of trees, and a summary of the grown forest. 2 to print more verbose output on processing of individual nodes. Default: 0. Due to the way %dopar% handles the output of the tree-growing iterations, you may not see the verbose output in some GUIs like RStudio. For best results, run R from the command line in order to see all the verbose output.

Value

The object supplied as the forest argument, with additional trees grown.

Methods

signature(forest = "bigcforest") Grow more trees in a classification random forest.

References


Examples

```
# Classify cars in the Cars93 data set by type (Compact, Large, Midsize, Small, Sporty, or Van).

# Load data.
data(Cars93, package="MASS")x <- Cars93y <- Cars93$Type

# Select variables with which to train model.
vars <- c(4:22)

# Run model, grow 30 trees.
forest <- bigrf(x, y, ntree=30L, varselect=vars, cachepath=NULL)

# Grow 10 more trees.
forest <- grow(forest, x, ntree=10L)
```

interactions-methods  Compute Variable Interactions

Description

Compute the interactions between variables, using an experimental method described by Breiman and Cutler. Variables m and n interact if a split on variable m in a tree makes a split on n either systematically less possible or more possible.
Usage

```r
## S4 method for signature 'bigcforest'
interactions(forest)
```

Arguments

- `forest`: A random forest of class "bigcforest".

Value

A symmetrical matrix of interactions between variables. A large positive number indicates that a split on one variable inhibits a split on the other variable, and conversely. This could indicate that the two variables are strongly correlated.

Methods

- `signature(forest = "bigcforest")`: Compute variable interactions for a classification random forest.

References


Examples

```r
# Classify cars in the Cars93 data set by type (Compact, Large, Midsize, Small, Sporty, or Van).

# Load data.
data(Cars93, package="MASS")
x <- Cars93
y <- Cars93$Type

# Select variables with which to train model.
vars <- c(4:22)

# Run model, grow 30 trees.
forest <- bigrf(x, y, ntree=30L, varselect=vars, cachepath=NULL)

# Calculate variable interactions.
inter <- interactions(forest)
```
**Merge Two Random Forests**

**Description**

Merge two random forests into one. This is useful, for example, for building random forests in parallel on multiple machines, then combining them into a single forest.

**Usage**

```r
## S4 method for signature 'bigcforest,bigcforest'
merge(x, y)
```

**Arguments**

- `x` A random forest of class "bigcforest".
- `y` A random forest of class "bigcforest", built using the same data sources and forest-building parameters as `x`.

**Details**

These methods copy all the trees from `y` into `x`, and calculates the error statistics and confusion matrices of the merged forest.

**Value**

The merged forest of the same class as the input arguments.

**Methods**

```r
signature(x = "bigcforest", y = "bigcforest") Merges the classification random forests `x` and `y`.
```

**References**


**Examples**

```r
# Classify cars in the Cars93 data set by type (Compact, Large, Midsize, Small, Sporty, or Van).

# Load data.
data(Cars93, package="MASS")
x <- Cars93
y <- Cars93$Type
```
outliers-methods

# Select variables with which to train model.
vars <- c(4:22)

# Run model, grow 15 trees.
forest <- bigrfc(x, y, ntree=15L, varselect=vars, cachepath=NULL)

# Build a second forest.
forest2 <- bigrfc(x, y, ntree=10L, varselect=vars, cachepath=NULL)

# Merge the two forests.
big.forest <- merge(forest, forest2)

---

### outliers-methods

**Compute Outlier Scores**

**Description**

Compute outlier scores for each class of examples used to train a random forest. Outliers are defined as examples whose proximities to other examples in the same class are small.

**Usage**

```r
## S4 method for signature 'bigcforest'
outliers(forest, trace=0L)
```

**Arguments**

- `forest`: A random forest of class "bigcforest".
- `trace`: 0 for no verbose output. 1 to print verbose output. Default: 0.

**Value**

A numeric vector containing the outlier scores for each training example. Higher scores indicate greater dissimilarity from other training examples in the same class.

**Methods**

- `signature(forest = "bigcforest")` Compute outlier scores for a classification random forest.

**References**


Examples

# Classify cars in the Cars93 data set by type (Compact, Large,
# Midsize, Small, Sporty, or Van).

# Load data.
data(Cars93, package="MASS")
x <- Cars93
y <- Cars93$Type

# Select variables with which to train model.
vars <- c(4:22)

# Run model, grow 30 trees.
forest <- bigrfc(x, y, ntree=30L, varselect=vars, cachepath=NULL)

# Calculate proximity matrix and scaling co-ordinates, and plot
# them.
prox <- proximities(forest, cachepath=NULL)
scale <- scaling(prox)
plot(scale, col=as.integer(y) + 2, pch=as.integer(y) + 2)

# Calculate outlier scores, and circle the top 20% percent of
# them in red.
outscores <- outliers(forest)
points(scale[outscores > quantile(outscores, probs=0.8), ],
    col=2, pch=1, cex=1.5)

predict-methods

Predict Classes of Test Examples

Description

Predict the classes of a set of test examples using a random forest.

Usage

## S4 method for signature 'bigcforest'
predict(object, x, y=NULL, printerrfreq=10L,
    printclserr=TRUE, cachepath=tempdir(), trace=0L)

Arguments

object A random forest of class "bigcforest".
x A big.matrix, matrix or data.frame of predictor variables. If a matrix or
data.frame is specified, it will be converted into a big.matrix for computa-
tion.
y An integer or factor vector of response variables. Test errors will be calculated
only if y is specified.
predict-methods

printerrfreq  An integer, specifying how often error estimates should be printed to the screen. Default: error estimates will be printed every 10 trees.

printclserr  TRUE for error estimates for individual classes to be printed, in addition to the overall error estimates. Default: TRUE.

cachepath  Path to folder where data caches used in building the forest can be stored. If NULL, then the big.matrix’s will be created in memory, with no disk caching; this will be suitable for small data sets. Default: tempdir()

trace  0 for no verbose output. 1 to print verbose output on prediction by trees, and a summary of the predictions. Default: 0. Due to the way %dopar% handles output, you may not see the verbose output in some GUIs like RStudio. For best results, run R from the command line in order to see all the verbose output.

Details
These methods copy all the trees from y into x, and calculates the error statistics and confusion matrices of the merged forest.

Value
An object of class "bigcprediction" containing the prediction results.

Methods
signature(object = "bigcforest") Predict classes of a set of test examples using a classification random forest.

References

Examples
# Classify cars in the Cars93 data set by type (Compact, Large, # Midsize, Small, Sporty, or Van).

# Load data.
data(Cars93, package="MASS")
x <- Cars93
y <- Cars93$Type

# Select variables with which to train model.
vars <- c(4:22)

# Run model, grow 30 trees on the first 60 examples.
forest <- bigrfc(x[1:60, ], y[1:60], ntree=30L, vselect=vars, cachepath=NULL)

# Get predictions for the remaining examples.
predictions <- predict(forest, x[-(1:60), ], y[-(1:60)], cachepath=NULL)
prototypes-methods

Compute Class Prototypes

Description
Compute the prototypes for each class in the training set, which provide a picture of how each variable relates to the classification. They are useful representations of a "typical" example of each class.

Usage
```r
## S4 method for signature 'bigforest,bigrfprox'
prototypes(forest, prox, nprot=1L, x=NULL, reuse.cache=FALSE, trace=0L)
```

Arguments
- `forest` A random forest of class "bigforest".
- `prox` A proximity matrix of class "bigrfprox".
- `nprot` The number of prototypes to compute for each class. Default: 1.
- `x` A `big.matrix`, matrix or `data.frame` of predictor variables. The data must not have changed, otherwise unexpected modelling results may occur. If a matrix or `data.frame` is specified, it will be converted into a `big.matrix` for computation. Optional if `reuse.cache` is `TRUE`.
- `reuse.cache` `TRUE` to reuse disk caches of the `big.matrix x` from the initial building of the random forest, which may significantly reduce initialization time for large data sets. If `TRUE`, the user must ensure that the files ‘x’ and ‘x.desc’ in `forest@ cachepath` have not been modified or deleted.
- `trace` 0 for no verbose output. 1 to print verbose output. Default: 0.

Details
Prototypes are computed using proximities, as follows. For the first prototype for class c, find the example i with the largest number of class c examples among its k nearest neighbors. Among these examples, find the 25th, 50th and 75th percentiles of the numeric variables, and most frequent level of the categorical variables. For the second prototype, the procedure is repeated, considering only examples that are not among the k examples used to compute the first prototype, and so on.

Value
A list with the following components:
- `nprotfound`: Number of prototypes found for each class.
- `clustersize`: `forest@ynclass by nprot` matrix indicating the number of examples used to compute each prototype.
prot: forest@ynclass by nprot by length(forest@varselect) by 3 array containing the raw prototype values. For numeric variables, the prototypes are represented by the medians, with the 25th and 75th percentiles given as estimates of the prototype stability. For categorical variables, the values are the most frequent level.

prot.std: forest@ynclass by nprot by length(forest@varselect) by 3 array containing standardized prototype values. Prototype values for numeric variables are subtracted by the 5th percentile, then divided by the difference between the 95th and 5th percentile. Prototype values for categorical variables are divided by the number of levels in that variable.

levelsfreq: List of length(length(forest@varselect) containing, for each categorical variable v, an forest@ynclass by nprot by forest@varnlevels[v] array that indicate the frequency of levels used to compute the prototype level. These are useful for estimating prototype stability for categorical variables.

Methods

signature(forest = "bigcforest", prox = "bigrfprox") Compute prototypes for a classification random forest.

References


Examples

# Classify cars in the Cars93 data set by type (Compact, Large,
# Midsize, Small, Sporty, or Van).

# Load data.
data(Cars93, package="MASS")
x <- Cars93
y <- Cars93$Type

# Select variables with which to train model.
vars <- c(4:22)

# Run model, grow 30 trees.
forest <- bigrfc(x, y, ntree=30L, varselect=vars, cachepath=NULL)

# Calculate proximity matrix.
prox <- proximities(forest, cachepath=NULL)

# Compute prototypes.
prot <- prototypes(forest, prox, x=x)

# Plot first prototypes, using one colour for each class.
plot(seq_along(vars), prot$prot std[1, 1, 2], type="l", col=1,
ylim=c(min(prot$prot std[, 1, 2]), max(prot$prot std[, 1, 2])))
for (i in 2:length(levels(y))) {
proximities-methods

Compute Proximity Matrix

Description

Compute the proximity matrix for a random forest, for the nnearest most proximate examples to each training example.

Usage

```r
## S4 method for signature 'bigcforest'
proximities(forest, nnearest=forest@nexamples, cachepath=tempdir(), trace=0L)
```

Arguments

- **forest**: A random forest of class "bigcforest".
- **nnearest**: The number of most proximate examples for which to compute proximity measures for each training example. Setting this to a smaller number will speed up computation of scaling co-ordinates. Default: `forest@nexamples`.
- **cachepath**: Path to folder where the proximity matrix can be stored. If NULL, then the big.matrix's will be created in memory with no disk caching, which would be suitable for small data sets. If the user wishes to reuse the cached files, it is suggested that a folder other than `tempdir()` is used, as the operating system may automatically delete any cache files in `tempdir()`. Default: `tempdir()`.
- **trace**: 0 for no verbose output. 1 to print verbose output. 2 to print even more verbose output on processing of each tree and example. Default: 0.

Value

An object of class "bigrfprox" containing the proximity matrix.

Methods

signature(forest = "bigcforest") Compute the proximity matrix for a classification random forest.
scaling-methods

References


Examples

```r
# Classify cars in the Cars93 data set by type (Compact, Large,
# Midsize, Small, Sporty, or Van).

# Load data.
data(Cars93, package="MASS")
x <- Cars93
y <- Cars93$Type

# Select variables with which to train model.
vars <- c(4:22)

# Run model, grow 30 trees.
forest <- bigrfc(x, y, ntree=30L, varselect=vars, cachepath=NULL)

# Calculate proximity matrix.
prox <- proximities(forest, cachepath=NULL)
```

---

**scaling-methods**

Compute Metric Scaling Co-ordinates

Description

Compute metric scaling co-ordinates for training examples in a random forest, based on the proximity matrix generated by `proximities`. Scaling co-ordinates are useful for visualising the data.

Usage

```r
## S4 method for signature 'bigrfprox'
scaling(prox, nscale=2L, trace=0L)
```

Arguments

- `prox`: A proximity matrix of class "bigrfprox".
- `nscale`: The number of scaling co-ordinates to compute. Typically, the first two or three scaling co-ordinates are the most useful. Default: 2L.
- `trace`: 0 for no verbose output. 1 to print verbose output. 2 to print even more verbose output on the progress of finding scaling coordinates. Default: 0.
Value

A matrix containing the scaling co-ordinates for each example, where the $i$th column contains the $i$th scaling co-ordinates.

Methods

signature(prox = "bigrfprox") Compute metric scaling coordinates for a random forest.

References


Examples

# Classify cars in the Cars93 data set by type (Compact, Large, Midsize, Small, Sporty, or Van).

# Load data.
data(Cars93, package="MASS")
x <- Cars93
y <- Cars93$Type

# Select variables with which to train model.
vars <- c(4:22)

# Run model, grow 30 trees.
forest <- bigrfc(x, y, ntree=30L, varselect=vars, cachepath=NULL)

# Calculate proximity matrix.
prox <- proximities(forest, cachepath=NULL)

# Calculate scaling co-ordinates.
scale <- scaling(prox, 3L)

# Plot the 1st vs 2nd scaling co-ordinates.
plot(scale[, 1], scale[, 2], col=as.integer(y), pch=as.integer(y))

# Plot the 1st vs 3rd scaling co-ordinates.
plot(scale[, 1], scale[, 3], col=as.integer(y), pch=as.integer(y))

# Plot the 2nd vs 3rd scaling co-ordinates.
plot(scale[, 2], scale[, 3], col=as.integer(y), pch=as.integer(y))
Compute Variable Importance

Description

Compute variable importance based on out-of-bag estimates. For each tree in the forest, the predictions of the out-of-bag examples are recorded. Then, a variable v is randomly permuted in the out-of-bag examples, and the tree is used to classify the out-of-bag examples again. The difference in votes for the correct class in the original data and the permuted data is used to calculate the variable importance for variable v. This process is then repeated for all variables.

Usage

```R
## S4 method for signature 'bigcforest'
varimp(forest, x=NULL, impbyexample=FALSE,
      reuse.cache=FALSE, trace=0L)
```

Arguments

- `forest` A random forest of class "bigcforest".
- `x` A `big.matrix`, `matrix` or `data.frame` of predictor variables. The data must not have changed, otherwise unexpected modelling results may occur. If a `matrix` or `data.frame` is specified, it will be converted into a `big.matrix` for computation. Optional if `reuse.cache` is `TRUE`.
- `impbyexample` A logical indicating whether to compute the variable importance for each out-of-bag example.
- `reuse.cache` `TRUE` to reuse disk caches of the `big.matrix x` from the initial building of the random forest, which may significantly reduce initialization time for large data sets. If `TRUE`, the user must ensure that the files ‘x’ and ‘x.desc’ in `forest@cachepath` have not been modified or deleted.
- `trace` 0 for no verbose output. 1 to print verbose output. Default: 0.

Value

A list with the following components:

- `importance`: Importance of each variable, which is the number of votes for the correct class in the out-of-bag examples with variable v permuted subtracted from the number of votes for the correct class in the original out-of-bag examples, averaged over all trees in the forest.
- `importanceNex`: Importance of each variable for each out-of-bag example.
- `zscore`: Z-score of each variable, computed by dividing the raw variable importance score by the standard error.
- `significance`: Significance level of each variable importance, computed by applying the complementary error function on the z-score.
Methods

signature(forest = "bigcforest") Compute variable importance for a classification random forest.

References


See Also

fastimp

Examples

# Classify cars in the Cars93 data set by type (Compact, Large, Midsize, Small, Sporty, or Van).

# Load data.
data(Cars93, package="MASS")
x <- Cars93
y <- Cars93$Type

# Select variables with which to train model.
vars <- c(4:22)

# Run model, grow 30 trees.
forest <- bigrfc(x, y, ntree=30L, varselect=vars, cachepath=NULL)

# Calculate variable importance, including those for each out-of-bag example.
importance <- varimp(forest, x, impbyexample=TRUE)
Index

*Topic classes
  bigcforest-class, 4
  bigcprediction-class, 6
  bigctree-class, 7
  bigrfprox-class, 10

*Topic methods
  fastimp-methods, 11
  grow-methods, 13
  interactions-methods, 14
  merge-methods, 16
  outliers-methods, 17
  predict-methods, 18
  prototypes-methods, 20
  proximities-methods, 22
  scaling-methods, 23
  varimp-methods, 25

*Topic models
  bigcforest-class, 4
  bigctree-class, 7
  bigrf-package, 2
  bigrfc, 8
  grow-methods, 13

*Topic package
  bigrf-package, 2

*Topic tree
  bigcforest-class, 4
  bigctree-class, 7
  bigrf-package, 2
  bigrfc, 8
  grow-methods, 13

%dopar%, 9, 14, 19

big.matrix, 2, 4, 8, 10, 12, 13, 18, 20, 25
bigcforest, 3, 7, 9, 11, 13, 15–18, 20, 22, 25
bigcforest-class, 4
bigcprediction, 19
bigcprediction-class, 6
bigctree, 3, 4, 9
bigctree-class, 7
bigmemory, 8

bigrf (bigrf-package), 2
bigrf-package, 2
bigrfc, 3, 4, 7, 8
bigrfprox, 20, 22, 23
bigrfprox-class, 10
cforest, 4, 9
data.frameRowLabels, 7
doParalleli, 3
fastimp, 6, 26
fastimp (fastimp-methods), 11
fastimp, bigcforest-method (fastimp-methods), 11
fastimp-methods, 11
foreach, 2, 3, 8
generateSyntheticClass, 12
grow, 3, 5
grow (grow-methods), 13
grow, bigcforest-method (grow-methods), 13
grow-methods, 13
integer, 7
interactions, 6
interactions (interactions-methods), 14
interactions, bigcforest-method (interactions-methods), 14
interactions-methods, 14
list, 5
merge, 4, 5
merge (merge-methods), 16
merge, bigcforest, bigcforest-method (merge-methods), 16
merge-methods, 16
numeric, 7

outliers (outliers-methods), 17
outliers.bigcforest-method
  (outliers-methods), 17
outliers-methods, 17

predict, 5, 6
predict (predict-methods), 18
predict.bigcforest-method
  (predict-methods), 18
predict-methods, 18
prototypes, 6, 10
prototypes (prototypes-methods), 20
prototypes.bigcforest.bigrfprox-method
  (prototypes-methods), 20
prototypes-methods, 20
proximities, 6, 10, 23
proximities (proximities-methods), 22
proximities.bigcforest-method
  (proximities-methods), 22
proximities-methods, 22

randomForest, 4, 9

scaling, 10
scaling (scaling-methods), 23
scaling.bigrfprox-method
  (scaling-methods), 23
scaling-methods, 23
show.bigcforest-method
  (bigcforest-class), 4
show.bigcprediction-method
  (bigcprediction-class), 6
summary.bigcforest-method
  (bigcforest-class), 4
summary.bigcprediction-method
  (bigcprediction-class), 6

varimp, 5, 9, 11
varimp (varimp-methods), 25
varimp.bigcforest-method
  (varimp-methods), 25
varimp-methods, 25
vector, 5, 7