

Package ‘yaImpute’

August 6, 2009

Version 1.0-10

Date 5 Aug 2009

Title yaImpute: An R Package for k-NN Imputation

Author Nicholas L. Crookston <ncrookston@fs.fed.us>, Andrew O. Finley <afinley@stat.umn.edu>

Maintainer Nicholas L. Crookston <ncrookston@fs.fed.us>

Depends R (>= 2.7.0)

Suggests vegan, randomForest, gam, fastICA, sp

Description Performs popular nearest neighbor routines for imputation

License GPL (>= 2)

Repository CRAN

Date/Publication 2009-08-06 10:56:09

R topics documented:

addXlevels	2
ann	3
AsciiGridImpute	6
compare.yai	11
cor.yai	12
errorStats	13
foruse	15
impute.yai	16
MoscowMtStJoe	18
mostused	21
newtargets	22
notablyDistant	23
plot.compare.yai	25
plot.yai	25
print.yai	26

rmsd.yai	27
TallyLake	28
unionDataJoin	29
vars	31
whatsMax	31
yai	32
yaiRFsummary	36
yaiVarImp	37

Index	39
--------------	-----------

addXlevels	<i>Adds xlevels to randomForest objects</i>
------------	---

Description

This function adds `xlevels` (see [lm](#)) to `randomForest` objects. Function `AsciiGridImpute` will check the levels on the input maps to insure that only those used in the fitting are used in the predictions.

Note that this function is not needed for objects built using `randomForest` version 4.5-21 or later.

Usage

```
addXlevels(object, origDataFrame)
```

Arguments

`object` an object built by `randomForest`
`origDataFrame` the data frame used in the `randomForest` fit.

Value

An object of class `randomForest` with `xlevels` appended.

Author(s)

Nicholas L. Crookston <ncrookston@fs.fed.us>
 Andrew O. Finley <finleya@msu.edu>

See Also

[yai](#) and [AsciiGridPredict](#)

Examples

```
if (require(randomForest))
{
  data(iris)
  rf = randomForest(x=iris[,2:5],y=iris[,1])
  new = addXlevels(rf,iris)
  print(new$xlevels)
  predict(new)
}
```

ann

Approximate nearest neighbor search routines

Description

Given a set of reference data points S , `ann` constructs a kd-tree or box-decomposition tree (bd-tree) for efficient k -nearest neighbor searches.

Usage

```
ann(ref, target, k=1, eps=0.0, tree.type="kd",
    search.type="standard", bucket.size=1, split.rule="sl_midpt",
    shrink.rule="simple", verbose=TRUE, ...)
```

Arguments

<code>ref</code>	an $n \times d$ matrix containing the reference point set S . Each row in <code>ref</code> corresponds to a point in d -dimensional space.
<code>target</code>	an $m \times d$ matrix containing the points for which k nearest neighbor reference points are sought.
<code>k</code>	defines the number of nearest neighbors to find. The default is $k=1$.
<code>eps</code>	the i^{th} nearest neighbor is at most $(1+eps)$ from true i^{th} nearest neighbor, where $eps \geq 0$. Specifically, the true (not squared) difference between the true i^{th} and the approximation of the i^{th} point is a factor of $(1+eps)$. The default value of $eps=0$ is an exact search.
<code>tree.type</code>	the data structures kd-tree or bd-tree as quoted key words <i>kd</i> and <i>bd</i> , respectively. A brute force search can be specified with the quoted key word <i>brute</i> . If <i>brute</i> is specified, then all subsequent arguments are ignored. The default is the kd-tree.
<code>search.type</code>	either standard or priority search in the kd-tree or bd-tree, specified by quoted key words <i>standard</i> and <i>priority</i> , respectively. The default is the standard search.
<code>bucket.size</code>	the maximum number of reference points in the leaf nodes. The default is 1.
<code>split.rule</code>	is the strategy for the recursive splitting of those nodes with more points than the bucket size. The splitting rule applies to both the kd-tree and bd-tree. Splitting rule options are the quoted key words:

`standard` - standard kd-tree
`midpt` - midpoint
`fair` - fair-split
`sl_midpt` - sliding-midpoint (default)
`sl{fair}` - fair-split rule

See supporting documentation, reference below, for a thorough description and discussion of these splitting rules.

`shrink.rule` applies only to the bd-tree and is an additional strategy (beyond the splitting rule) for the recursive partitioning of nodes. This argument is ignored if `tree.type` is specified as `kd`. Shrinking rule options are quoted key words:

`none` - equivalent to the kd-tree
`simple` - simple shrink (default)
`centroid` - centroid shrink

See supporting documentation, reference below, for a thorough description and discussion of these shrinking rules.

`verbose` if true, search progress is printed to the screen.
`...` currently no additional arguments.

Details

The `ann` function calls portions of the Approximate Nearest Neighbor Library, written by David M. Mount. All of the `ann` function arguments are detailed in the ANN Programming Manual found at <http://www.cs.umd.edu/~mount/ANN>.

Value

An object of class `ann`, which is a list with some or all of the following tags:

`knnIndexDist` an $m \times 2k$ matrix. Each row corresponds to a target point in `target` and columns `1:k` hold the `ref` matrix row indices of the nearest neighbors, such that column `1` index holds the `ref` matrix row index for the first nearest neighbor and column `k` is the k^{th} nearest neighbor index. Columns `k + 1:2k` hold the Euclidean distance from the target to each of the k nearest neighbors indexed in columns `1:k`.

`searchTime` total search time, not including data structure construction, etc.

`k` as defined in the `ann` function call.

`eps` as defined in the `ann` function call.

`tree.type` as defined in the `ann` function call.

`search.type` as defined in the `ann` function call.

`bucket.size` as defined in the `ann` function call.

`split.rule` as defined in the `ann` function call.

`shrink.rule` as defined in the `ann` function call.

Author(s)

Andrew O. Finley <finleya@msu.edu>

Examples

```
## Make a couple of bivariate normal classes
rmvn <- function(n, mu=0, V = matrix(1))
{
  p <- length(mu)
  if(any(is.na(match(dim(V), p))))
    stop("Dimension problem!")
  D <- chol(V)
  matrix(rnorm(n*p), ncol=p) %*% D + rep(mu, rep(n, p))
}

m <- 10000

## Class 1.
mu.1 <- c(20, 40)
V.1 <- matrix(c(-5, 1, 0, 5), 2, 2); V.1 <- V.1%*%t(V.1)
c.1 <- cbind(rmvn(m, mu.1, V.1), rep(1, m))

## Class 2.
mu.2 <- c(30, 60)
V.2 <- matrix(c(4, 2, 0, 2), 2, 2); V.2 <- V.2%*%t(V.2)
c.2 <- cbind(rmvn(m, mu.2, V.2), rep(2, m))

## Class 3.
mu.3 <- c(15, 60)
V.3 <- matrix(c(5, 5, 0, 5), 2, 2); V.3 <- V.3%*%t(V.3)
c.3 <- cbind(rmvn(m, mu.3, V.3), rep(3, m))

c.all <- rbind(c.1, c.2, c.3)
max.x <- max(c.all[,1]); min.x <- min(c.all[,1])
max.y <- max(c.all[,2]); min.y <- min(c.all[,2])

## Check them out.
plot(c.1[,1], c.1[,2], xlim=c(min.x, max.x), ylim=c(min.y, max.y),
     pch=19, cex=0.5,
     col="blue", xlab="Variable 1", ylab="Variable 2")
points(c.2[,1], c.2[,2], pch=19, cex=0.5, col="green")
points(c.3[,1], c.3[,2], pch=19, cex=0.5, col="red")

## Take a reference sample.
n <- 2000
ref <- c.all[sample(1:nrow(c.all), n),]

## Compare search times
k <- 10
## Do a simple brute force search.
```

```

brute <- ann(ref=ref[,1:2], target=c.all[,1:2],
            tree.type="brute", k=k, verbose=FALSE)
print(brute$searchTime)

## Do an exact kd-tree search.
kd.exact <- ann(ref=ref[,1:2], target=c.all[,1:2],
              tree.type="kd", k=k, verbose=FALSE)
print(kd.exact$searchTime)

## Do an approximate kd-tree search.
kd.approx <- ann(ref=ref[,1:2], target=c.all[,1:2],
               tree.type="kd", k=k, eps=100, verbose=FALSE)
print(kd.approx$searchTime)

## Takes too long to calculate for this many targets.
## Compare overall accuracy of the exact vs. approximate search
##knn.mode <- function(knn.indx, ref){
##  x <- ref[knn.indx,]
##  as.numeric(names(sort(as.matrix(table(x))[,1],
##                       decreasing=TRUE))[1])
##}

```

AsciiGridImpute *Imputes/Predicts data for Ascii Grid maps*

Description

AsciiGridImpute finds nearest neighbor *reference* observations for each point in the input grid maps and outputs maps of selected Y-variables in a set of output grid maps.

AsciiGridPredict applies a predict function to each point in the input grid maps and outputs maps of the prediction(s) in one or more output grid maps (see Details).

One row of the each grid maps is read and processed at a time thereby avoiding the need to build huge objects in R that would be necessary if all the rows of all the maps were processed together.

Usage

```

AsciiGridImpute(object, xfiles, outfiles, xtypes=NULL, ancillaryData=NULL,
               ann=NULL, lon=NULL, lat=NULL, rows=NULL, cols=NULL,
               nodata=NULL, myPredFunc=NULL, ...)

```

```

AsciiGridPredict(object, xfiles, outfiles, xtypes=NULL, lon=NULL, lat=NULL,
                rows=NULL, cols=NULL, nodata=NULL, myPredFunc=NULL, ...)

```

Arguments

`object` An object of class `yai`, any other object for which a `predict` function is defined, or an object that is passed to a predict function you define using argument `myPredFunc`. See Details.

xfiles	A list of input file names where there is one grid file for each X-variable. List elements must be given the same names as the X-variables they correspond with and there must be one file for each X-variable used when <code>object</code> was built.
outfiles	One of these two forms: <ol style="list-style-type: none"> (1) A file name that is understood to correspond to the single prediction returned by the generic <code>predict</code> function related to <code>object</code> or returned by <code>myPredFunc</code>. This form only applies to <code>AsciiGridPredict</code>, when the <code>object</code> is not class <code>yai</code>. (2) A list of output file names where there is one grid file for each <i>desired</i> output variable. While there may be many variables predicted for <code>object</code>, only those for which an output grid is desired need to be specified. Note that some predict functions return data frames, some return a single vector, and often what is returned depends on the value of arguments passed to <code>predict</code>. In addition to names of the predicted variables, the following two special names can be coded when the object class is <code>yai</code>: For <code>distance="filename"</code> a map of the distances is output and if <code>useid="filename"</code> a map of useid's is output. When the predict function returns a vector, an additional special name of <code>predict="filename"</code> can be used.
xtypes	A list of data type names that corresponds exactly to data type of the maps listed in <code>xfiles</code> . Each value can be one of: "logical", "integer", "numeric", "character". If NULL, or if a type is missing for a member of <code>xfiles</code> , type "numeric" is used. See Details if you used factors as predictors.
ancillaryData	A data frame of Y-variables that may not have been used in the original call to <code>yai</code> . There must be one row for each reference observation, no missing data, and row names must match those used in the original reference observations.
ann	if NULL, the value is taken from <code>object</code> . When TRUE, <code>ann</code> is used to find neighbors, and when FALSE a slow exact search is used (ignored for when method <code>randomForest</code> is used when the original <code>yai</code> object was created).
lon	if NULL, the value of <code>cols</code> is used. Otherwise, a 2-element vector given the range of longitudes (horizontal distance) desired for the output.
lat	if NULL, the value of <code>rows</code> is used. Otherwise, a 2-element vector given the range of latitudes (vertical distance) desired for the output.
rows	if NULL, all rows from the input grids are used. Otherwise, <code>rows</code> is a 2-element vector given the rows desired for the output. If the second element is greater than the number of rows, the header value <code>YLLCORNER</code> is adjusted accordingly. Ignored if <code>lon</code> is specified.
cols	if NULL, all columns from the input grids are used. Otherwise, <code>cols</code> is a 2-element vector given the columns desired for the output. If the first element is greater than one, the header value <code>XLLCORNER</code> is adjusted accordingly. Ignored if <code>lat</code> is specified.
nodata	the <code>NODATA_VALUE</code> for the output. If NULL, the value is taken from the input grids.

`myPredFunc` called to predict output using the `object` and `newdata` from the `xfiles`. Two arguments are passed to this function, the first is the value of `object` and the second is a data frame of the new predictor variables created for each row of data from your input maps. If `NULL`, the generic `predict` function is called for `object`.

`...` passed to `myPredFunc` or `predict`.

Details

The input maps are assumed to be AsciiGrid maps with 6-line headers containing the following tags: `NCOLS`, `NROWS`, `XLLCORNER`, `YLLCORNER`, `CELLSIZE` and `NODATA_VALUE` (case insensitive). The headers should be identical, a warning is issued if they are not. It is critical that `NODATA_VALUE` is the same on all input maps.

The function builds data frames from the input maps one row at a time and builds predictions using those data frames as *newdata*. Each row of the input maps is processed in sequence so that the entire maps are not stored in memory. The function works by opening all the input and reads one line (row) at a time from each. The output file(s) are created one line at a time as the input maps are processed.

Use `AsciiGridImpute` for objects built with `yai`, otherwise use `AsciiGridPredict`. When `AsciiGridPredict` is used, the following rules apply. First, when `myPredFunc` is not null it is called with the arguments `object`, `newdata`, `...` where the new data is the data frame built from the input maps, otherwise the generic `predict` function is called with these same arguments. When `object` and `myPredFunc` are both `NULL` a copy `newdata` used as the prediction. This is useful when `lat`, `lon`, `rows`, or `cols` are used in to subset the maps.

The `NODATA_VALUE` is output for every `NODATA_VALUE` found on any grid cell on any one of the input maps (the `predict` function is not called for these grid cells). `NODATA_VALUE` is also output for any grid cell where the `predict` function returns an `NA`. If factors are used as `X`-variables in `object`, the levels found the map data are checked against those used in building the `object`. If new levels are found, the corresponding output map grid point is set to `NODATA_VALUE`; the `predict` function is not called for these cells as most `predict` functions will fail in these circumstances. Checking on factors depends on `object` containing a meaningful member named `xlevels`, as done objects model objects produced by `lm`. Note that objects produced by `randomForest` version 4.5-19 and prior do not contain `xlevels`; use function `addXlevels` to add the necessary element if one or more `factors` are used.

AsciiGrid maps do not contain character data, only numbers. The numbers in the maps are matched the `xlevels` by subscript (the first entry in a level corresponds to the numeric value 1 in the AsciiGrid maps, the second to the number 2 and so on). Care must be taken by the user to insure that the coding scheme used in building the maps is identical to that used in building the `object`. See Value for information on how you can check the matching of these codes.

Value

An `invisible` list containing the following named elements:

`unexpectedNAs`

A data frame listing the map row numbers and the number of `NA` values generated by the `predict` function for each row. If none are generated for a row the row is not reported, if none are generated for any rows, the data frame is `NULL`.

- `illegalLevels` A data frame listing levels found in the maps that were not found in the `xlevels` for the `object`. The row names are the illegal levels, the column names are the variable names, and the values are the number of grid cells where the illegal levels were found.
- `outputLegend` A data frame showing the relationship between levels in the output maps and those found in `object`. The row names are level index values, the column names are variable names, and the values are the levels. NULL if no factors are output.
- `inputLegend` A data frame showing the relationship between levels found in the input maps and those found in `object`. The row names are level index values (this function assumes they correspond to numeric values on the maps), the column names are variable names, and the values are the levels. NULL if no factors are input. This information is basically the same as that in `xlevels`.

Author(s)

Nicholas L. Crookston <ncrookston@fs.fed.us>
 Andrew O. Finley <finleya@msu.edu>

See Also

[yai](#), [impute](#), and [newtargets](#)

Examples

```
## These commands write new files to your working directory

# Use the iris data
data(iris)

# Section 1: Imagine that the iris are planted in a planting bed.
# The following set of commands create AsciiGrid map
# files for four attributes to illustrate the planting layout.

# Change species from a character factor to numeric (the sp classes
# can not handle character data).

sLen <- matrix(iris[,1],10,15)
sWid <- matrix(iris[,2],10,15)
pLen <- matrix(iris[,3],10,15)
pWid <- matrix(iris[,4],10,15)
spcd <- matrix(as.numeric(iris[,5]),10,15)

# Make maps of each variable.

header = c("NCOLS 15", "NROWS 10", "XLLCORNER 1", "YLLCORNER 1",
          "CELLSIZE 1", "NODATA_VALUE -9999")
cat(file="slen.txt", header, sep="\n")
cat(file="swid.txt", header, sep="\n")
```

```

cat (file="plen.txt",header,sep="\n")
cat (file="pwid.txt",header,sep="\n")
cat (file="spcd.txt",header,sep="\n")

write.table(sLen,file="slen.txt",append=TRUE,col.names=FALSE,
            row.names=FALSE)
write.table(sWid,file="swid.txt",append=TRUE,col.names=FALSE,
            row.names=FALSE)
write.table(pLen,file="plen.txt",append=TRUE,col.names=FALSE,
            row.names=FALSE)
write.table(pWid,file="pwid.txt",append=TRUE,col.names=FALSE,
            row.names=FALSE)
write.table(spcd,file="spcd.txt",append=TRUE,col.names=FALSE,
            row.names=FALSE)

# Section 2: Create functions to predict species

# set the random number seed so that example results are consistant
# normally, leave out this command
set.seed(12345)

# sample the data
refs <- sample(rownames(iris),50)
y <- data.frame(Species=iris[refs,5],row.names=rownames(iris[refs,]))

# build a yai imputation for the reference data.
rfNN <- yai(x=iris[refs,1:4],y=y,method="randomForest")

# make lists of input and output map files.

xfiles <- list(Sepal.Length="slen.txt",Sepal.Width="swid.txt",
              Petal.Length="plen.txt",Petal.Width="pwid.txt")
outfiles1 <- list(distance="dist.txt",Species="spOutrfNN.txt")

# map the imputation-based predictions for the input maps
AsciiGridImpute(rfNN,xfiles,outfiles1,ancillaryData=iris)

# build a randomForest predictor
rf <- randomForest(x=iris[refs,1:4],y=iris[refs,5])

# map the predictions for the input maps
outfiles2 <- list(predict="spOutrf.txt")
AsciiGridPredict(rf,xfiles,outfiles2,xtypes=NULL,rows=NULL)

# read the species maps and plot them using the sp package classes
if (require(sp)) {
  spOrig <- read.asciigrid("spcd.txt")
  sprfNN <- read.asciigrid("spOutrfNN.txt")
  sprf <- read.asciigrid("spOutrf.txt")
  dist <- read.asciigrid("dist.txt")

  par(mfcol=c(2,2))
  image(spOrig,col=c(1,2,3))

```

```

    title("Original")
    image(sprfNN, col=c(1, 2, 3))
    title("Using Predict")
    image(sprf, col=c(1, 2, 3))
    title("Using Impute")
    image(dist)
    title("Neighbor Distances")
  }

```

 compare.yai

Compares different k-NN solutions

Description

Provides a convenient display of the root mean square differences (see [rmsd.yai](#)) or correlations (see [cor.yai](#)) between observed and imputed values for each of several imputations. Each column of the returned data frame corresponds to an imputation result and each row corresponds to a variable.

Usage

```
compare.yai(..., ancillaryData=NULL, vars=NULL, method="rmsd")
```

Arguments

<code>...</code>	a list of objects created by yai or impute.yai that you wish to compare.
<code>ancillaryData</code>	a data frame that defines new variables, passed to impute.yai .
<code>vars</code>	a list of variable names you want to include; if <code>NULL</code> all available variables are included.
<code>method</code>	when <i>rmsd</i> is specified, the comparison is based on root mean square differences between observed and imputed, and when <i>cor</i> is specified, the comparison is based on correlations between observed and imputed.

Value

A data.frame of class `c("compare.yai", "data.frame")`, where the columns are the names of the `...`-arguments and the rows are a union of variable names. `NA`'s are returned when the variables are factors.

Author(s)

Nicholas L. Crookston (ncrookston@fs.fed.us)
Andrew O. Finley (finleya@msu.edu)

See Also

[yai](#), [plot.compare.yai](#), [impute.yai](#), [rmsd.yai](#)

Examples

```
require(yaImpute)

data(iris)

# form some test data
refs=sample(rownames(iris),50)
x <- iris[,1:2]      # Sepal.Length Sepal.Width
y <- iris[refs,3:4] # Petal.Length Petal.Width

# build yai objects using 2 methods
msn <- yai(x=x,y=y)
mal <- yai(x=x,y=y,method="mahalanobis")

# compare the y variables
compare.yai(msn,mal)

# compare the all variables in iris
compare.yai(msn,mal,ancillaryData=iris) # Species is a factor, no comparison is made
```

cor.yai

Correlation between observed and imputed

Description

Computes the correlation between observed and imputed values for each observation that has both.

Usage

```
cor.yai (object,vars=NULL,...)
```

Arguments

object	an object created by yai or impute.yai .
vars	a list of variables names you want to include, if NULL all available variables are included.
...	passed to called methods (not currently used)

Details

The correlations are computed using [cor.yai](#). For data imputation, such correlations are likely not appropriate; a warning message is issued when this function is used (Stage and Crookston 2007).

Value

A data frame with the row names as vars and the column as cor.

Author(s)

Nicholas L. Crookston <ncrookston@fs.fed.us>

Andrew O. Finley <finleya@msu.edu>

References

Stage, A.R.; Crookston, N.L. (2007). Partitioning error components for accuracy-assessment of near neighbor methods of imputation. *For. Sci.* 53(1):62-72. http://www.fs.fed.us/rm/pubs_other/rmrs_2007_stage_a001.pdf

See Also

[yai](#), [impute.yai](#), [rmsd.yai](#)

errorStats

Compute error components of k-NN imputations

Description

Error properties of estimates derived from imputation differ from those of regression-based estimates because the two methods include a different mix of error components. This function computes a partitioning of error statistics as proposed by Stage and Crookston (2007).

Usage

```
errorStats(mahal, ..., scale=FALSE, pzero=0.1, plg=0.5, seeMethod="lm")
```

Arguments

mahal	An object of class yai computed with <code>method="mahalanobis"</code> .
...	Other objects of class yai for which statistics are desired. All objects should be for the same data and variables used for the first argument.
scale	When TRUE, the errors are scaled by their respective standard deviations.
pzero	The lower tail p-value used to pick <i>reference</i> observations that are zero distance from each other (used to compute <code>rmmsd0</code>).
plg	The upper tail p-value used to pick <i>reference</i> observations that are substantially distant from each other (used to compute <code>rmsdlg</code>).
seeMethod	Method used to compute SEE: <code>seeMethod="lm"</code> uses lm and <code>seeMethod="gam"</code> uses gam . In both cases, the model formula is a simple linear combination of the X-variables.

Details

See http://www.fs.fed.us/rm/pubs_other/rmrs_2007_stage_a001.pdf

Value

A list that contains several data frames. The column names of each are a combination of the name of the object used to compute the statistics and the name of the statistic. The rownames correspond to the Y-variables from the first argument. The data frame names are as follows:

common	statistics used to compute other statistics.
name of first argument	error statistics for the first <code>yai</code> object.
names of ... arguments	error statistics for each of the remaining <code>yai</code> objects, if any.
see	standard error of estimate for individual regressions fit for corresponding Y-variables.
rmmsd0	root mean square difference for imputations based on <code>method="mahalanobis"</code> (always based on the first argument to the function).
mlf	square root of the model lack of fit: $\sqrt{see^2 - (rmmsd0^2/2)}$.
rmsd	root mean square error.
rmsdlg	root mean square error of the observations with larger distances.
sei	standard error of imputation $\sqrt{rmsd^2 - (rmmsd0^2/2)}$.
dstc	distance component: $\sqrt{rmsd^2 - rmmsd0^2}$.

Note that unlike Stage and Crookston (2007), all statistics reported here are in the natural units, not squared units.

Author(s)

Nicholas L. Crookston <ncrookston@fs.fed.us>
 Albert R. Stage <astage@moscow.com>

References

Stage, A.R.; Crookston, N.L. (2007). Partitioning error components for accuracy-assessment of near neighbor methods of imputation. *For. Sci.* 53(1):62-72. <http://forest.moscowfs1.wsu.edu/gems/StagePartitioningFS.pdf>

See Also

[yai](#), [TallyLake](#)

Examples

```
require (yaImpute)

data(TallyLake)

diag(cov(TallyLake[,1:8])) # see col A in Table 3 in Stage and Crookston

mal=yai(x=TallyLake[,9:29],y=TallyLake[,1:8],ann=FALSE,
        noTrgs=TRUE,method="mahalanobis")

msn=yai(x=TallyLake[,9:29],y=TallyLake[,1:8],ann=FALSE,
        noTrgs=TRUE,method="msn")

# variable "see" for "mal" matches col B (when squared and scaled)
# other columns don't match exactly as Stage and Crookston used different
# software to compute values

errorStats(mal,msn)
```

foruse

Report a complete imputation

Description

Provides a matrix of all observations with the reference observation identification best used to represent it, followed by the distance.

Usage

```
foruse(object, kth=NULL)
```

Arguments

object	an object created by yai
kth	when NULL, the best pick is reported, otherwise the kth neighbor is picked.

Author(s)

Nicholas L. Crookston <ncrookston@fs.fed.us>
Andrew O. Finley <finleya@msu.edu>

Examples

```
require(yaImpute)

data(iris)

# form some test data
refs=sample(rownames(iris),50)
x <- iris[,1:3]      # Sepal.Length Sepal.Width Petal.Length
y <- iris[refs,4:5] # Petal.Width Species

# build a yai object using mahalanobis
mal <- yai(x=x,y=y,method="mahalanobis",k=3)

foruse(mal) # for references, use is equal to the rowname
foruse(mal,kth=1) # for references, use is an row to the kth reference.

# get all the choices:
cbind(foruse(mal),foruse(mal,kth=1),foruse(mal,kth=2),foruse(mal,kth=3))
```

impute.yai

Impute variables from references to targets

Description

Imputes the observation for variables from a *reference* observation to a *target* observation. Also, imputes a value for a *reference* from other *references*. This practice is useful for validation (see [yai](#)). Variables not available in the original data may be imputed using argument `ancillaryData`.

Usage

```
## S3 method for class 'yai':
impute(object, ancillaryData=NULL, method="closest",
        method.factor=method, k=NULL, vars=NULL,
        observed=TRUE, ...)
```

Arguments

<code>object</code>	an object of class yai .
<code>ancillaryData</code>	a data frame of variables that may not have been used in the original call to yai . There must be one row for each reference observation, no missing data, and row names must match those used in the reference observations.
<code>method</code>	the method used to compute the imputed values for continuous variables, as follows: <code>closest</code> : use the single neighbor that is closest (this is the default and is always used when $k=1$); <code>mean</code> : an average over the k neighbors is taken;

	dstWeighted: a weighted average is taken over the k neighbors where the weights are $1/(1+d)$.
method.factor	the method used to compute the imputed values for factors, as follows: closest: use the single neighbor that is closest (this is the default and is always used when $k=1$); mean: actually is the <i>mode</i> it is the factor level that occurs the most often among the k neighbors; dstWeighted: a <i>mode</i> where the count is the sum of the weights ($1/(1+d)$) rather than each having a weight of 1.
k	the number neighbors to use in averages, when NULL all present are used.
vars	a character vector of variables to impute, when NULL, the behaviour depends on the value of ancillaryData: when it is NULL, the Y-variables are imputed others all present in ancillaryData are imputed.
observed	when TRUE, columns are created for <i>observed</i> values (those from the <i>target</i> observations) as well as imputed values (those from the <i>reference</i> observations).
...	passed to other methods, currently not used.

Value

An object of class `impute.yai`, which is a data frame with rownames identifying observations and column names identifying variables. When `observed=TRUE` additional columns are created with a suffix of `.o`.

NA's fill columns of observed values when no corresponding value is known, as in the case for Y-variables from *target* observations.

Scale factors for each variable are returned as an attribute (see [attributes](#)).

Author(s)

Nicholas L. Crookston <ncrookston@fs.fed.us>
Andrew O. Finley <finleya@msu.edu>
Emilie Grossmann <Emilie.Grossmann@oregonstate.edu>

See Also

[yai](#)

Examples

```
require(yaImpute)

data(iris)

# form some test data
refs=sample(rownames(iris),50)
x <- iris[,1:3]      # Sepal.Length Sepal.Width Petal.Length
y <- iris[refs,4:5] # Petal.Width Species
```

```
# build a yai object using mahalanobis
mal <- yai(x=x,y=y,method="mahalanobis")

# output a data frame of observed and imputed values
# of all variables and observations.

impute(mal)
malImp=impute(mal,ancillaryData=iris)
plot(malImp)
```

MoscowMtStJoe

Moscow Mountain and St. Joe Woodlands (Idaho, USA) Tree and Li-DAR Data

Description

Data used to compare the utility of discrete-return light detection and ranging (LiDAR) data and multispectral satellite imagery, and their integration, for modeling and mapping basal area and tree density across two diverse coniferous forest landscapes in north-central Idaho, USA.

Usage

```
data(MoscowMtStJoe)
```

Format

A data frame with 165 rows and 64 columns:

Ground based measurements of trees:

ABGR_BA	Basal area (m^2/ha) of ABGR
ABLA_BA	Basal area (m^2/ha) of ABLA
ACGL_BA	Basal area (m^2/ha) of ACGL
BEOC_BA	Basal area (m^2/ha) of BEOC
LAOC_BA	Basal area (m^2/ha) of LAOC
PICO_BA	Basal area (m^2/ha) of PICO
PIEN_BA	Basal area (m^2/ha) of PIEN
PIMO_BA	Basal area (m^2/ha) of PIMO
PIPO_BA	Basal area (m^2/ha) of PIPO
POBA_BA	Basal area (m^2/ha) of POBA
POTR_BA	Basal area (m^2/ha) of POTR
PSME_BA	Basal area (m^2/ha) of PSME
SAEX_BA	Basal area (m^2/ha) of SAEX

THPL_BA	Basal area (m^2/ha) of THPL
TSHE_BA	Basal area (m^2/ha) of TSHE
TSME_BA	Basal area (m^2/ha) of TSME
UNKN_BA	Basal area (m^2/ha) of unknown species
Total_BA	Basal area (m^2/ha) total over all species
ABGR_TD	Trees per ha of ABGR
ABLA_TD	Trees per ha of ABLA
ACGL_TD	Trees per ha of ACGL
BEOC_TD	Trees per ha of BEOC
LAOC_TD	Trees per ha of LAOC
PICO_TD	Trees per ha of PICO
PIEN_TD	Trees per ha of PIEN
PIMO_TD	Trees per ha of PIMO
PIPO_TD	Trees per ha of PIPO
POBA_TD	Trees per ha of POBA
POTR_TD	Trees per ha of POTR
PSME_TD	Trees per ha of PSME
SAEX_TD	Trees per ha of SAEX
THPL_TD	Trees per ha of THPL
TSHE_TD	Trees per ha of TSHE
TSME_TD	Trees per ha of TSME
UNKN_TD	Trees per ha of unknown species
Total_TD	Trees per ha total over all species

Geographic Location, Slope and Aspect:

EASTING	UTM (Zone 11) easting at plot center
NORTHING	UTM (Zone 11) northing at plot center
ELEVATION	Mean elevation (m) above sea level over plot
SLPMEAN	Mean slope (percent) over plot
ASPMEAN	Mean aspect (degrees) over plot

Advanced Land Imager (ALI):

B1MEAN	Mean of 30 m ALI band 1 pixels intersecting plot
B2MEAN	Mean of 30 m ALI band 2 pixels intersecting plot
B3MEAN	Mean of 30 m ALI band 3 pixels intersecting plot
B4MEAN	Mean of 30 m ALI band 4 pixels intersecting plot
B5MEAN	Mean of 30 m ALI band 5 pixels intersecting plot
B6MEAN	Mean of 30 m ALI band 6 pixels intersecting plot

B7MEAN Mean of 30 m ALI band 7 pixels intersecting plot
B8MEAN Mean of 30 m ALI band 8 pixels intersecting plot
B9MEAN Mean of 30 m ALI band 9 pixels intersecting plot
PANMEAN Mean of 10 m PAN band pixels intersecting plot
PANSTD Standard deviation of 10 m PAN band pixels intersecting plot

LiDAR Intensity:

INTMEAN Mean of 2 m intensity pixels intersecting plot
INTSTD Standard deviation of 2 m intensity pixels intersecting plot
INTMIN Minimum of 2 m intensity pixels intersecting plot
INTMAX Maximum of 2 m intensity pixels intersecting plot

LiDAR Height:

HTMEAN Mean of 6 m height pixels intersecting plot
HTSTD Standard deviation of 6 m height pixels intersecting plot
HTMIN Minimum of 6 m height pixels intersecting plot
HTMAX Maximum of 6 m height pixels intersecting plot

LiDAR Canopy Cover:

CCMEAN Mean of 6 m canopy cover pixels intersecting plot
CCSTD Standard deviation of 6 m canopy cover pixels intersecting plot
CCMIN Minimum of 6 m canopy cover pixels intersecting plot
CCMAX Maximum of 6 m canopy cover pixels intersecting plot

Source

Dr. Andrew T. Hudak
USDA Forest Service
Rocky Mountain Research Station
1221 South Main
Moscow, Idaho, USA 83843

References

Hudak, A.T.; Crookston, N.L.; Evans, J.S.; Falkowski, M.J.; Smith, A.M.S.; Gessler, P.E.; Morgan, P. (2006). Regression modeling and mapping of coniferous forest basal area and tree density from discrete-return lidar and multispectral satellite data. *Can. J. Remote Sensing*. 32(2):126-138.
<http://www.treesearch.fs.fed.us/pubs/24612>

`mostused`*Tabulate references most often used in imputation*

Description

Provides a matrix of reference observations that are used most often as sources of imputation and a column of the counts. The observations are listed in sorted order, most often used first.

Usage

```
mostused(object, n=20, kth=NULL)
```

Arguments

<code>object</code>	(1) a data frame created by <code>foruse</code> , or (2) an object created by <code>yai</code> in which case <code>foruse</code> is called automatically.
<code>n</code>	the number of mostused in sorted order.
<code>kth</code>	passed to <code>foruse</code> , if called.

Author(s)

Nicholas L. Crookston <ncrookston@fs.fed.us>
Andrew O. Finley <finleya@msu.edu>

See Also

`yai`

Examples

```
require(yaImpute)

data(iris)

# form some test data
refs=sample(rownames(iris),50)
x <- iris[,1:3]      # Sepal.Length Sepal.Width Petal.Length
y <- iris[refs,4:5] # Petal.Width Species

# build a yai object using mahalanobis
mal <- yai(x=x,y=y,method="mahalanobis")

mostused(mal,kth=1)
```

 newtargets

Finds K nearest neighbors for new target observations

Description

Finds nearest neighbor *reference* observations for a given set of *target* observations using an established (see [yai](#)) object. Intended use is to facilitate breaking up large imputation problems (see [AsciiGridImpute](#)).

Usage

```
newtargets(object, newdata, ann=NULL)
```

Arguments

object	an object of class yai .
newdata	a data frame or matrix of new <i>targets</i> for which neighbors are found.
ann	if NULL, the value is taken from <code>object</code> . When TRUE <code>ann</code> is used to find neighbors, and when FALSE a slow exact search is used.

Value

An object of class `yai`, which is a copy of the first argument with the following elements replaced:

call	the call.
obsDropped	a list of the row names for observations dropped for various reasons (missing data).
trgRows	a list of the row names for target observations as a subset of all observations.
xall	the X-variables for all observations.
neiDstTrgs	a data frame of distances between a target (identified by its row name) and the k references. There are k columns.
neiIdsTrgs	A data frame of reference identifications that correspond to <code>neiDstTrgs</code> .
neiDstRefs	set NULL as if <code>noRefs=TRUE</code> in the original call to yai .
neiIdsRefs	set NULL as if <code>noRefs=TRUE</code> in the original call to yai .
ann	the value of the <code>ann</code> argument.

Author(s)

Nicholas L. Crookston <ncrookston@fs.fed.us>
 Andrew O. Finley <finleya@msu.edu>

See Also

[yai](#)

Examples

```

require (yaImpute)

data(iris)

# set the random number seed so that example results are consistent
# normally, leave out this command
set.seed(12345)

# form some test data
refs=sample(rownames(iris),50) # just the reference observations
x <- iris[refs,1:3] # Sepal.Length Sepal.Width Petal.Length
y <- iris[refs,4:5] # Petal.Width Species

# build a yai object using mahalanobis
mal <- yai(x=x,y=y,method="mahalanobis")

# get imputations for the target observations (not references)
malNew <- newtargets(mal,iris[!(rownames(iris) %in% rownames(x)),])

# output a data frame of observed and imputed values for
# the observations that are not in the original yai object

impute(malNew,vars=yvars(malNew))

# in this example, Y is not specified (not required for mahalanobis).
mal2 <- yai(x=x,method="mahalanobis")
identical(foruse(mal),foruse(mal2))

# here, method randomForest's unsupervised classification is used (no Y).
rf <- yai(x=x,method="randomForest")
# now get imputations for the targets in the iris data (those that are
# not references.
rfNew <- newtargets(rf,iris[!(rownames(iris) %in% rownames(x)),])

```

notablyDistant *Find notably distant targets*

Description

Notably distant *targets* are those with relatively large distances from the closest *reference* observation. A suitable *threshold* is used to detect large distances.

Usage

```
notablyDistant(object, kth=1, threshold=NULL, p=0.01)
```

Arguments

object	an object of class <code>yai</code> .
kth	the kth neighbor is used.
threshold	the threshold distance that identifies <i>notably</i> large distances between observations.
p	the percentile point in the distribution of distances used to compute the threshold (only used when <i>threshold</i> is NULL).

Details

When `threshold` is NULL, the function computes one by assuming the distances follow the lognormal distribution, unless the method used to find neighbors is `randomForest`, in which case the distances are assumed to follow the beta distribution. A specified `p` value is used to compute the threshold.

Value

List of two data frames that contain 1) the *references* that are notably distant from other *references*, 2) the *targets* that are notably distant from the *references*, and 3) the *threshold* used.

Author(s)

Nicholas L. Crookston <ncrookston@fs.fed.us>
Andrew O. Finley <finleya@msu.edu>

See Also

[yai](#)

Examples

```
data(iris)

# form some test data
refs=sample(rownames(iris),50)
x <- iris[,1:3]      # Sepal.Length Sepal.Width Petal.Length
y <- iris[refs,4:5] # Petal.Width Species

# build an msn run, first build dummy variables for species.

sp1 <- as.integer(iris$Species=="setosa")
sp2 <- as.integer(iris$Species=="versicolor")
y2 <- data.frame(cbind(iris[,4], sp1, sp2), row.names=rownames(iris))
y2 <- y2[refs,]

names(y2) <- c("Petal.Width", "Sp1", "Sp2")

msn <- yai(x=x, y=y2, method="msn")

notablyDistant(msn)
```

plot.compare.yai *Plots a compare.yai object*

Description

Provides a matrix of plots for objects created by `compare.yai`.

Usage

```
## S3 method for class 'compare.yai':  
plot(x, pointColor=1, lineColor=2, ...)
```

Arguments

`x` a data frame created by `compare.yai`.
`pointColor` the color used for the points.
`lineColor` the color of the 1:1 line.
`...` passed to plot functions.

Author(s)

Nicholas L. Crookston <ncrookston@fs.fed.us>
Andrew O. Finley <finleya@msu.edu>

See Also

`yai`, `compare.yai`, `impute.yai`, `rmsd.yai`

plot.yai *Plot observed verses imputed data*

Description

Provides a matrix of plots of observed verses imputed values for variables in an object created by `impute.yai`, which are of class `c("impute.yai", "data.frame")`.

Usage

```
## S3 method for class 'yai':  
plot(x, vars=NULL, pointColor=1, lineColor=2, spineColor=NULL, residual=FALSE, ...)
```

Arguments

x	(1) a data frame created by <code>impute.yai</code> , or (2) an object created by <code>yai</code> .
vars	a list of variable names you want to include, if NULL all available Y-variables are included.
pointColor	a color vector for the xy plots (continuous variables).
lineColor	a color 1:1 lines in xy plots.
spineColor	a color vector for the spine plots (factors), one value per level.
residual	plots in a residual format (observed-imputed over imputed).
...	passed to called functions.

Author(s)

Nicholas L. Crookston <ncrookston@fs.fed.us>
 Andrew O. Finley <finleya@msu.edu>

Examples

```
require(yaImpute)

data(iris)

# form some test data
refs=sample(rownames(iris),50)
x <- iris[,1:3]      # Sepal.Length Sepal.Width Petal.Length
y <- iris[refs,4:5] # Petal.Width Species

mal <- yai(x=x,y=y,method="mahalanobis")
malImp=impute(mal,newdata=iris)
plot(malImp)
```

```
print.yai
```

Print a summary of a yai object

Description

Provides a summary of a `yai` object, showing the call and essential data customized for each method used.

Usage

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

Arguments

x an object of class `yai`.
 ... not used

Author(s)

Nicholas L. Crookston <ncrookston@fs.fed.us>
 Andrew O. Finley <finleya@msu.edu>

rmsd.yai

*Root Mean Square Difference between observed and imputed***Description**

Computes the root mean square difference (RMSD) between observed and imputed values for each observation that has both. RMSD is computationally like RMSE, but they differ in interpretation. By default the RMSD values are scaled by the standard deviation of the respective variable.

Usage

```
rmsd.yai (object, vars=NULL, scale=FALSE, ...)
```

Arguments

object an object created by `yai` or `impute.yai`
 vars a list of variable names you want to include, if `NULL` all available variables are included
 scale when `TRUE`, the values are scaled (see details)
 ... passed to called methods, very useful for passing argument `ancillaryData` to function `impute.yai`

Details

By default, RMSD is computed using standard formula for its related statistic, RMSE. When `scale=TRUE`, RMSD is divided by the standard deviation of the observed data. The standard deviation is computed over the original list of *reference* observations and is therefore not influenced by the number of times a given reference is used for imputation.

Value

A data frame with the row names as `vars` and the column as `rmsd`. When `scale=TRUE`, the column name is `rmsdS`.

Author(s)

Nicholas L. Crookston <ncrookston@fs.fed.us>
 Andrew O. Finley <finleya@msu.edu>

See Also

[yai](#), [impute.yai](#) and <http://www.jstatsoft.org/v23/i10>.

TallyLake

Tally Lake, Flathead National Forest, Montana, USA

Description

Polygon-based reference data used by Stage and Crookston (2007) to demonstrate partitioning of error components and related statistics. Observations are summaries of data collected on forest stands (poygons).

Usage

```
data(TallyLake)
```

Format

A data frame with 847 rows and 29 columns:

Ground based measurements of trees (Y-variables):

TopHt	Height of tallest trees (ft)
LnVolL	Log of the volume ($ft^3/acre$) of western larch
LnVolDF	Log of the volume ($ft^3/acre$) of Douglas-fir
LnVolLP	Log of the volume ($ft^3/acre$) of lodgepole pine
LnVolES	Log of the volume ($ft^3/acre$) of Engelmann spruce
LnVolAF	Log of the volume ($ft^3/acre$) of alpine fir
LnVolPP	Log of the volume ($ft^3/acre$) of ponderosa pine
CCover	Canopy cover (percent)

Geographic Location, Slope, and Aspect (X-variables):

utmX	UTM easting at plot center
utmY	UTM northing at plot center
elevm	Mean elevation (ft) above sea level over plot
eevsqrd	$(elevm - 1600)^2$
slopem	Mean slope (percent) over plot
slpcosaspm	Mean of slope (proportion) times the cosine of aspect (see Stage (1976) for description of this transformation)
slpsinaspm	Mean of slope (proportion) times the sine of aspect

Additional X-variables:

ctim	Mean of slope curvature over pixels in stand
tmb1m	Mean of LandSat band 1 over pixels in stand
tmb2m	Mean of LandSat band 2 over pixels in stand
tmb3m	Mean of LandSat band 3 over pixels in stand
tmb4m	Mean of LandSat band 4 over pixels in stand
tmb5m	Mean of LandSat band 5 over pixels in stand
tmb6m	Mean of LandSat band 6 over pixels in stand
durm	Mean of light duration over pixels in stand
inson	Mean of solar insolation over pixels in stand
msavim	Mean of AVI for pixels in stand
ndvim	Mean of NDVI for pixels in stand
crvm	Mean of slope curvature for pixels in stand
tancrvm	Mean of tangent curvature for pixels in stand
tancrvsd	Standard deviation of tangent curvature for pixels in stand

Source

USDA Forest Service

References

Stage, A.R.; Crookston, N.L. 2007. Partitioning error components for accuracy-assessment of near neighbor methods of imputation. *For. Sci.* 53(1):62-72 <http://www.treesearch.fs.fed.us/pubs/28385>

Stage, A.R. (1976). An expression for the effect of aspect, slope, and habitat type on tree growth. *For. Sci.* 22(4):457-460.

unionDataJoin

Combines data from several sources

Description

Takes any combination of several data frames or matrices and creates a new data frame. The rows are defined by a union of all row names in the arguments, and the columns are defined by a union of all column names in the arguments. The data are loaded into this new frame where column and row names match the individual inputs. Duplicates are tolerated with the last one specified being the one kept. NAs are returned for combinations of rows and columns where no data exist. Factors are processed as necessary.

Usage

```
unionDataJoin(..., warn=TRUE)
```

Arguments

... a list of data frames, matrices, or any combination.
warn when TRUE, warn when a column name is found in more than one data source.

Value

A data frame.

Author(s)

Nicholas L. Crookston <ncrookston@fs.fed.us>
Andrew O. Finley <finleya@msu.edu>

Examples

```
require(yaImpute)

d1=data.frame(x1=c("a", "b", "c", "d", "e", "f"))
d2=data.frame(x1=as.character(seq(1,4)), row.names=seq(5,8))
d3=data.frame(x2=seq(1:10))

# note the levels
levels(d1$x1)
# [1] "a" "b" "c" "d" "e" "f"

levels(d2$x1)
# [1] "1" "2" "3" "4"

all=unionDataJoin(d1,d2,d3, warn=FALSE)
all
#       x1 x2
# 1     a  1
# 2     b  2
# 3     c  3
# 4     d  4
# 5     1  5
# 6     2  6
# 7     3  7
# 8     4  8
# 9 <NA>  9
# 10 <NA> 10

levels(all$x1)
# [1] "1" "2" "3" "4" "a" "b" "c" "d"
```

vars *List variables in a yai object*

Description

Provides a character vector, or a list of character vectors of all the variables in a [yai](#) object, just the X-variables (`xvars`), or just the Y-variables (`yvars`).

Usage

```
vars(object)
xvars(object)
yvars(object)
```

Arguments

`object` an object created by [yai](#).

Value

`yvars` A character vector of Y-variables.
`xvars` A character vector of X-variables.
`vars` A list of both vectors.

Author(s)

Nicholas L. Crookston <ncrookston@fs.fed.us>
 Andrew O. Finley <finleya@msu.edu>

See Also

[yai](#)

what'sMax *Find maximum column for each row*

Description

For each row, the function identifies the column that has the maximum value. The function returns a data frame with two columns: the first is the column name corresponding to the column of maximum value and the second is the correspond maximum. The first column is converted to a factor.

If the maximum is zero, the maximum column is identified as "zero".

If there are over `nbig` factors in column 1, the maximum values that are less than the largest are combined and identified as "other".

Intended use is to transform community ecology data for use in [yai](#) where method is *randomForest*.

Usage

```
whatsMax(x, nbig=30)
```

Arguments

`x` a data frame or matrix of numeric values.
`nbig` see description—the maximum number of factors, the remainder called 'other'.

Value

A data frame.

Author(s)

Nicholas L. Crookston (ncrookston@fs.fed.us)
 Andrew O. Finley (finleya@msu.edu)

Examples

```
data(MoscowMtStJoe)

# get the basal area by species columns
yba <- MoscowMtStJoe[,1:17]

# for each row, pick the species that has the max basal area
# create "other" for those not in the top 7.

ybaB <- whatsMax(yba, nbig=7)
levels(ybaB[,1])
```

yai

Find K nearest neighbors

Description

Given a set of observations, `yai` 1) separates the observations into *reference* and *target* observations, 2) applies the specified method to project X-variables into a Euclidean space (not always, see argument `method`), and 3) finds the *k*-nearest neighbors within the reference observations and between the reference and target observations. An alternative method using randomForest classification and regression trees is provided for steps 2 and 3. *Target* observations are those with values for X-variables and not for Y-variables, while *reference* observations are those with no missing values for X- and Y-variables (see Details for the exception).

Usage

```
yai(x=NULL, y=NULL, data=NULL, k=1, noTrgs=FALSE, noRefs=FALSE,
    nVec=NULL, pVal=.05, method="msn", ann=TRUE, mtry=NULL, ntree=500,
    rfMode="buildClasses")
```

Arguments

<code>x</code>	1) a matrix or data frame containing the X-variables for all observations with row names are the identification for the observations, or 2) a one-sided formula defining the X-variables as a linear formula. If a formula is coded for <code>x</code> , one must be used for <code>y</code> as well, if needed.
<code>y</code>	1) a matrix or data frame containing the Y-variables for the reference observations, or 2) a one-sided formula defining the Y-variables as a linear formula.
<code>data</code>	when <code>x</code> and <code>y</code> are formulas, then <code>data</code> is a data frame or matrix that contains all the variables with row names are the identification for the observations. The observations are split by <code>yai</code> into two sets.
<code>k</code>	the number of nearest neighbors; default is 1.
<code>noTrgs</code>	when TRUE, skip finding neighbors for target observations.
<code>noRefs</code>	when TRUE, skip finding neighbors for reference observations.
<code>nVec</code>	number of canonical vectors to use (methods <code>msn</code> and <code>msn2</code>), or number of independent of X-variables reference data when method <code>mahalanobis</code> . When NULL, the number is set by the function.
<code>pVal</code>	significant level for canonical vectors, used when method is <code>msn</code> or <code>msn2</code> .
<code>method</code>	is the strategy finding neighbors; the options are the quoted key words (see details): <ul style="list-style-type: none"> <code>euclidean</code> distance is computed in a normalized X space. <code>raw</code> like euclidean, except no normalization is done. <code>mahalanobis</code> distance is computed in its namesakes space. <ul style="list-style-type: none"> <code>ica</code> like mahalanobis, but based on <i>Independent Component Analysis</i> using package <code>fastICA</code>. <code>msn</code> distance is computed in a projected canonical space. <code>msn2</code> like <code>msn</code>, but with variance weighting (canonical regression rather than correlation). <code>gnn</code> distance is computed using a projected ordination of Xs found using canonical correspondence analysis (<code>cca</code> from package vegan). <code>randomForest</code> distance is one minus the proportion of randomForest trees where a target observation is in the same terminal node as a reference observation (see <code>randomForest</code>).
<code>ann</code>	TRUE if <code>ann</code> is used to find neighbors, FALSE if a slow search is used.
<code>mtry</code>	the number of X-variables picked at random when method is <code>randomForest</code> , see <code>randomForest</code> , default is <code>sqrt(number of X-variables)</code> .
<code>ntree</code>	the number of classification and regression trees when method is <code>randomForest</code> . When more than one Y-variable is used, the trees are divided among the variables. Alternatively, <code>ntree</code> can be a vector of values corresponding to each Y-variable.
<code>rfMode</code>	when <code>buildClasses</code> and method is <code>randomForest</code> , continuous variables are internally converted to classes forcing <code>randomForest</code> to build classification trees for the variable. Otherwise, regression trees are built if your version of <code>randomForest</code> is newer than 4.5-18.

Details

See the paper at <http://www.jstatsoft.org/v23/i10> (it includes examples).

The following information is in addition to the content in the papers.

You need not have any Y-variables to run `yai` for the following methods: `euclidean`, `raw`, `mahalanobis`, `ica`, and `randomForest` (in which case unsupervised classification is performed). However, normally `yai` classifies *reference* observations as those with no missing values for X- and Y- variables and *target* observations are those with values for X- variables and missing data for Y-variables. When Y is NULL (there are no Y-variables), all the observations are considered *references*. See `newtargets` for an example of how to use `yai` in this situation.

Value

An object of class `yai`, which is a list with the following tags:

<code>call</code>	the call.
<code>yRefs</code> , <code>xRefs</code>	matrices of the X- and Y-variables for just the reference observations (unscaled). The scale factors are attached as attributes.
<code>obsDropped</code>	a list of the row names for observations dropped for various reasons (missing data).
<code>trgRows</code>	a list of the row names for target observations as a subset of all observations.
<code>xall</code>	the X-variables for all observations.
<code>cancor</code>	returned from <code>cancor</code> function when method <code>msn</code> or <code>msn2</code> is used (NULL otherwise).
<code>ccaVegan</code>	an object of class <code>cca</code> (from package vegan) when method <code>gnn</code> is used.
<code>ftest</code>	a list containing partial F statistics and a vector of $\text{Pr}>F$ (pgf) corresponding to the canonical correlation coefficients when method <code>msn</code> or <code>msn2</code> is used (NULL otherwise).
<code>yScale</code> , <code>xScale</code>	scale data used on <code>yRefs</code> and <code>xRefs</code> as needed.
<code>k</code>	the value of k .
<code>pVal</code>	as input; only used when method <code>msn</code> or <code>msn2</code> is used.
<code>projector</code>	NULL when not used. For methods <code>msn</code> , <code>msn2</code> , <code>gnn</code> and <code>mahalanobis</code> , this is a matrix that projects normalized X-variables into a space suitable for doing Eculidian distances.
<code>nVec</code>	number of canonical vectors used (methods <code>msn</code> and <code>msn2</code>), or number of independent X-variables in the reference data when method <code>mahalanobis</code> is used.
<code>method</code>	as input, the method used.
<code>ranForest</code>	a list of the forests if method <code>randomForest</code> is used. There is one forest for each Y-variable, or just one forest when there are no Y-variables.
<code>ICA</code>	a list of information from <code>fastICA</code> when method <code>ica</code> is used.
<code>ann</code>	the value of <code>ann</code> , TRUE when <code>ann</code> is used, FALSE otherwise.

xlevels NULL if no factors are used as predictors; otherwise a list of predictors that have factors and their levels (see [lm](#)).

neiDstTrgs a data frame of distances between a target (identified by its row name) and the k references. There are k columns.

neiIdsTrgs a data frame of reference identifications that correspond to neiDstTrgs.

neiDstRefs, neiIdsRefs counterparts for references.

Author(s)

Nicholas L. Crookston <ncrookston@fs.fed.us>
 Andrew O. Finley <finleya@msu.edu>

Examples

```
require (yaImpute)

data(iris)

# set the random number seed so that example results are consistent
# normally, leave out this command
set.seed(12345)

# form some test data, y's are defined only for reference
# observations.
refs=sample(rownames(iris),50)
x <- iris[,1:2]        # Sepal.Length Sepal.Width
y <- iris[refs,3:4]   # Petal.Length Petal.Width

# build yai objects using 2 methods
msn <- yai(x=x,y=y)
mal <- yai(x=x,y=y,method="mahalanobis")

# running the following examples will load packages vegan
# and randomForest, and is more complicated.

data(MoscowMtStJoe)

# convert polar slope and aspect measurements to cartesian
# (which is the same as Stage's (1976) transformation).

polar <- MoscowMtStJoe[,40:41]
polar[,1] <- polar[,1]*.01        # slope proportion
polar[,2] <- polar[,2]*(pi/180) # aspect radians
cartesian <- t(apply(polar,1,function (x)
                  {return (c(x[1]*cos(x[2]),x[1]*sin(x[2])) )}))
colnames(cartesian) <- c("xSlAsp","ySlAsp")
x <- cbind(MoscowMtStJoe[,37:39],cartesian,MoscowMtStJoe[,42:64])
y <- MoscowMtStJoe[,1:35]
```

```

mal <- yai(x=x, y=y, method="mahalanobis", k=1)
gnn <- yai(x=x, y=y, method="gnn", k=1)
msn <- yai(x=x, y=y, method="msn", k=1)

plot(mal, vars=yvars(mal)[1:16])

# reduce the plant community data for randomForest.
yba <- MoscowMtStJoe[,1:17]
ybaB <- whatsMax(yba, nbig=7) # see help on whatsMax

rf <- yai(x=x, y=ybaB, method="randomForest", k=1)

# build the imputations for the original y's
rforig <- impute(rf, ancillaryData=y)

# compare the results
compare.yai(mal, gnn, msn, rforig)
plot(compare.yai(mal, gnn, msn, rforig))

# build another randomForest case forcing regression
# to be used for continuous variables. The answers differ
# but one is set not clearly better than the other.

rf2 <- yai(x=x, y=ybaB, method="randomForest", rfMode="regression")
rforig2 <- impute(rf2, ancillaryData=y)
compare.yai(rforig2, rforig)

```

yaiRFsummary

Build Summary Data For Method RandomForest

Description

When method `randomforest` is used to build a `yai` object, the `randomForest` package computes several statistics. This function summarizes some of them, including the variable importance scores computed by function `yaiVarImp`.

Usage

```
yaiRFsummary(object, nTop=0)
```

Arguments

<code>object</code>	an object of class <code>yai</code> .
<code>nTop</code>	the <code>nTop</code> most important variables are plotted (returned).

Value

A list containing:

`forestAttributes`

a data frame reporting the error rates and other data from the `randomForest(s)`.

`scaledImportance`

the data frame computed by `yaiVarImp`.

Author(s)

Nicholas L. Crookston <ncrookston@fs.fed.us>

Andrew O. Finley <finleya@msu.edu>

See Also

`yai`, `yaiVarImp`

`yaiVarImp`

Replots or plots importance scores for method randomForest

Description

When method `randomForest` is used to build a `yai` object, the `randomForest` package computes variable importance scores. This function computes a composite of the scores and scales them. By default the scores are plotted and scores themselves are invisibly returned.

Usage

```
yaiVarImp(object, nTop=20, plot=TRUE, ...)
```

Arguments

`object` an object of class `yai`

`nTop` the `nTop` most important variables are plotted (returned); if NA or zero, all are returned

`plot` if FALSE, no plotting is done, but the scores are returned.

`...` passed to `boxplot` function.

Value

A data frame with the rows corresponding to the `randomForest` built for each *Y*-variable and the columns corresponding to the `nTop` most important *Y*-variables in sorted order.

Author(s)

Nicholas L. Crookston <ncrookston@fs.fed.us>

Andrew O. Finley <finleya@msu.edu>

See Also

[yai](#), [yaiRFsummary](#), [compare.yai](#)

Examples

```
data(MoscowMtStJoe)

# get the basal area by species columns
yba <- MoscowMtStJoe[,1:17]
ybaB <- whatsMax(yba,nbig=7) # see help on whatsMax

ba <- cbind(ybaB,TotalBA=MoscowMtStJoe[,18])
x <- MoscowMtStJoe[,37:64]
x <- x[,-(4:5)]
rf <- yai(x=x,y=ba,method="randomForest")

yaiVarImp(rf)

keep=colnames(yaiVarImp(rf,plot=FALSE,nTop=9))

newx <- x[,keep]
rf2 <- yai(x=newx,y=ba,method="randomForest")

yaiVarImp(rf2,col="gray")

compare.yai(rf,rf2)
```

Index

*Topic **datasets**

MoscowMtStJoe, 17
TallyLake, 27

*Topic **hplot**

plot.compare.yai, 24
plot.yai, 25

*Topic **misc**

addXlevels, 1
ann, 2
cor.yai, 12
impute.yai, 16
mostused, 20
notablyDistant, 23
unionDataJoin, 29
vars, 30
whatsMax, 31
yaiRFsummary, 36
yaiVarImp, 37

*Topic **multivariate**

addXlevels, 1
compare.yai, 10
cor.yai, 12
errorStats, 13
foruse, 15
impute.yai, 16
mostused, 20
newtargets, 21
notablyDistant, 23
rmsd.yai, 26
yai, 32
yaiRFsummary, 36
yaiVarImp, 37

*Topic **print**

print.yai, 26

*Topic **spatial**

AsciiGridImpute, 5

*Topic **tree**

yaiRFsummary, 36
yaiVarImp, 37

*Topic **utilities**

AsciiGridImpute, 5

addXlevels, 1, 8
ann, 2, 7, 21, 33, 34
AsciiGridImpute, 1, 5, 21
AsciiGridPredict, 2
AsciiGridPredict
(AsciiGridImpute), 5
attributes, 17

cca, 33
compare.yai, 10, 24, 25, 37
cor.yai, 10, 12, 12

errorStats, 13

factor, 8
fastICA, 33, 34
foruse, 15, 20

gam, 13

impute, 8
impute(impute.yai), 16
impute.yai, 11, 12, 16, 25, 27
invisible, 8

list, 6
lm, 1, 8, 13, 34

MoscowMtStJoe, 17
mostused, 20

newtargets, 8, 21, 33
notablyDistant, 23

plot.compare.yai, 11, 24
plot.impute.yai(plot.yai), 25
plot.yai, 25
predict, 6, 7

`print.yai`, 26

`randomForest`, 1, 2, 8, 33, 36, 37

`rmsd.yai`, 10–12, 25, 26

`summary.yai` (`print.yai`), 26

TallyLake, 14, 27

`unionDataJoin`, 29

`vars`, 30

`whatsMax`, 31

`xvars` (`vars`), 30

`yai`, 2, 6–8, 11–17, 20–27, 30, 31, 32, 36, 37

`yaImpute` (`yai`), 32

`yaiRFsummary`, 36, 37

`yaiVarImp`, 36, 37

`yvars` (`vars`), 30