Package ‘yaImpute’

November 3, 2022

Type Package
Title Nearest Neighbor Observation Imputation and Evaluation Tools
Version 1.0-33
Date 2022-10-24
Description Performs nearest neighbor-based imputation using one or more alternative approaches to processing multivariate data. These include methods based on canonical correlation: analysis, canonical correspondence analysis, and a multivariate adaptation of the random forest classification and regression techniques of Leo Breiman and Adele Cutler. Additional methods are also offered. The package includes functions for comparing the results from running alternative techniques, detecting imputation targets that are notably distant from reference observations, detecting and correcting for bias, bootstrapping and building ensemble imputations, and mapping results.
Depends R (>= 3.0.0)
Imports grDevices, graphics, stats, utils
Suggests vegan, ccaPP, randomForest, gam, fastICA, parallel, gower
Copyright ANN library is copyright University of Maryland and Sunil Arya and David Mount. See file COPYRIGHTS for details.
Maintainer Jeffrey S. Evans <jeffrey_evans@tnc.org>
License GPL (>= 2)
URL https://github.com/jeffreyevans/yaImpute
BugReports https://github.com/jeffreyevans/yaImpute/issues
NeedsCompilation yes
Repository CRAN
Encoding UTF-8
RoxygenNote 7.2.1
Author Jeffrey S. Evans [aut, cre] (<https://orcid.org/0000-0002-5533-7044>), Nicholas L. Crookston [aut], Andrew O. Finley [aut], John Coulston (Sunil Arya and David Mount for ANN) [ctb, com]

Date/Publication 2022-11-03 22:20:02 UTC
R topics documented:

ann | applyMask | AsciiGridImpute | bestVars | buildConsensus | compare.yai | cor.yai | correctBias | ensembleImpute | errorStats | foruse | grmsd | impute.yai | MoscowMtStJoe | mostused | newtargets | notablyDifferent | notablyDistant | plot.compare.yai | plot.notablyDifferent | plot.varSel | plot.yai | predict.yai | print.yai | rmsd.yai | TallyLake | unionDataJoin | vars | varSelection | whatsMax | yai | yaiRFsummary | yaiVarImp

Index

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.
ann

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

ref

an \( n \times d \) matrix containing the reference point set \( S \). Each row in \( \text{ref} \) corresponds to a point in \( d \)-dimensional space.

target

an \( m \times d \) matrix containing the points for which \( k \) nearest neighbor reference points are sought.

k

defines the number of nearest neighbors to find. The default is \( k=1 \).

eps

the \( i^{th} \) nearest neighbor is at most \((1+\text{eps})\) from true \( i^{th} \) nearest neighbor, where \( \text{eps} \geq 0 \). Specifically, the true (not squared) difference between the true \( i^{th} \) and the approximation of the \( i^{th} \) point is a factor of \((1+\text{eps})\). The default value of \( \text{eps}=0 \) is an exact search.

tree.type

the data structures kd-tree or bd-tree as quoted key words \( \text{kd} \) and \( \text{bd} \), respectively. A brute force search can be specified with the quoted key word \( \text{brute} \). If \( \text{brute} \) is specified, then all subsequent arguments are ignored. The default is the \( \text{kd} \)-tree.

search.type

either standard or priority search in the \( \text{kd} \)-tree or \( \text{bd} \)-tree, specified by quoted key words \( \text{standard} \) and \( \text{priority} \), respectively. The default is the standard search.

bucket.size

the maximum number of reference points in the leaf nodes. The default is 1.

split.rule

is the strategy for the recursive splitting of those nodes with more points than the bucket size. The splitting rule applies to both the \( \text{kd} \)-tree and \( \text{bd} \)-tree. Splitting rule options are the quoted key words:

- \text{standard} - standard \( \text{kd} \)-tree
- \text{midpt} - midpoint
- \text{fair} - fair-split
- \text{sl\_midpt} - sliding-midpoint (default)
- \text{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 \( \text{bd} \)-tree and is an additional strategy (beyond the splitting rule) for the recursive partitioning of nodes. This argument is ignored if \text{tree.type} is specified as \text{kd}. Shrinking rule options are quoted key words:

- \text{none} - equivalent to the \text{kd} tree
- \text{simple} - simple shrink (default)
- \text{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 https://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

```r
## 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])
}
##
applyMask

Removes neighbors that share (or not) group membership with targets.

Description

Some of the nearest neighbors found using yai or newtargets are removed using this function. This is possible when there are several reference observations for each target as is the case with \( k > 1 \). The function removes neighbor reference observations for a given target if the reference and target are in (a) the same group or (b) from different groups, depending on the method used. Group membership is identified for reference and target observations using two vectors, refGroups for references and trgGroups for targets. If the group membership code is the same for a reference and a target, then they are in the same group while different codes mean a lack of common group membership.

Usage

applyMask(object, refGroups=NULL, trgGroups=NULL, method="removeWhenCommon", k=1)

Arguments

object an object of class yai.
refGroups a vector, with length equal to the number of reference observations, of codes that indicate group membership.
trgGroups a vector, with length equal to the number of target observations, of codes that indicate group membership. The data type and coding scheme of refGroups and trgGroups must be the same.
method is the strategy used for removing neighbors from the object, as follows:
  - removeWhenCommon remove neighbors where the group membership of a target is the same as the group membership of the near neighbor reference (that is, keep near neighbors if they are not in the same group).
  - keepWhenCommon keep near neighbors only when the reference is in the same group as the target (that is, remove near neighbors if they are not in the same group).
k the number of nearest neighbors to keep.

Value

An object of class yai, that is a copy of the first argument with the following elements replaced:
call the call.
neiDstTrgs a matrix of distances between a target (identified by its row name) and the k references. There are k columns.
neiIdsTrgs a matrix of reference identifications that correspond to neiDstTrgs.
n eiDstRefs set NULL as if noRefs=TRUE in the original call to yai.
n eiIdsRefs set NULL as if noRefs=TRUE in the original call to yai.
applyMask

Arguments

- noRefs: set TRUE regardless of original value.
- k: the value of k.

Author(s)

Nicholas L. Crookston <ncrookston.fs@gmail.com>

Acknowledgment: This function was inspired by correspondence with Clara Anton Fernandez.

See Also

- `yai newtargets`

Examples

```r
require (yaImpute)

data(iris)

# build a base case, there are no targets,
# turn off getting references neighbors.
mal <- yai(x=iris[, -5], method="mahalanobis", noRefs = TRUE)

# create a new data, just a copy of the old with new row names.
iris2 <- iris
rownames(iris2) <- paste0("new.", rownames(iris))

# do an imputation with k=55
m55 <- newtargets(mal, newdata=iris2, k=55)

# get the 2 closest where the species codes don't match by
# removing neighbors when the ref group membership is
# in common with the target group membership (same species),
# thereby forcing neighbors to be from different species.

# in this case, the groups are species codes.
applyMask(m55, refGroups=iris$Species, trgGroups=iris2$Species, method="removeWhenCommon", k=2)

# get the 2 closest where the species codes do match by
# removing neighbors when the ref group membership is
# different than the target group membership (different species),
# thereby forcing neighbors to be from the same species (this
# is generally true anyway using the iris data).

applyMask(m55, iris$Species, trgGroups=iris2$Species, method="keepWhenCommon", k=2)
```
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 corresponding 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 corresponding output grid maps (see Details).

One row of each grid map 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 \texttt{yai}, any object for which a \texttt{predict} function is defined, or an object that is passed to a predict function you define using argument \texttt{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 object was built.

outfiles One of these two forms:

- (1) A file name that is understood to correspond to the single prediction returned by the generic \texttt{predict} function related to object or returned by \texttt{myPredFunc}. This form only applies to AsciiGridPredict, when the object is not class \texttt{yai}.

- (2) A \texttt{list} of output file names where there is one grid file for each \texttt{desired} output variable. While there may be many variables predicted for object, only those for which an output grid is desire 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 predict. In addition to names of the predicted variables, the following two special names can be coded when the object class is \texttt{yai}: For distance=“filename” a map of the distances is output and if userid=“filename” a map of integer indices to row numbers of the reference observations is output. When the predict function returns a vector, an additional special name of predict=“filename” can be used.
xtypes
A list of data type names that corresponds exactly to data type of the maps listed in xfiles. Each value can be one of: "logical", "integer", "numeric", "character". If NULL, or if a type is missing for a member of xfiles, 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 yai. 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 object. When TRUE, ann is used to find neighbors, and when FALSE a slow exact search is used (ignored for when method randomForest is used when the original yai object was created).

lon
if NULL, the value of cols is used. Otherwise, a 2-element vector given the range of longitudes (horizontal distance) desired for the output.

lat
if NULL, the value of rows 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, rows 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 YLLCORNER in the output is adjusted accordingly. Ignored if lon is specified.

cols
if NULL, all columns from the input grids are used. Otherwise, cols is a 2-element vector given the columns desired for the output. If the first element is greater than one, the header value XLLCORNER in the output is adjusted accordingly. Ignored if lat is specified.

nodata
the NODATA_VALUE for the output. If NULL, the value is taken from the input grids.

myPredFunc
called by AsciiGridPredict to predict output using the object and newdata from the xfiles. Two arguments are passed by AsciiGridPredict 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, predict, or impute.

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 for all input maps, 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 time as the input maps are processed.

Use AsciiGridImpute for objects builds 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 of `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 for objects produced by `lm`.

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 consistent with that in `xlevels`.

Author(s)

Nicholas L. Crookston <ncrookston.fs@gmail.com>

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)

# Create and change to a temp directory. You can delete these steps if you wish to keep the files in your working directory.
curdir <- getwd()
setwd(tempdir())
cat("Using working dir",getwd(),"\n")

# 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",
  useid="useindx.txt")

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

# read the ascigrids and get them ready to plot
spOrig <- t(as.matrix(read.table("spcd.txt",skip=6)))
sprfNN <- t(as.matrix(read.table("spOutrfNN.txt",skip=6)))
dist <- t(as.matrix(read.table("dist.txt",skip=6)))

# demonstrate the use of useid:
spViaUse <- read.table("useindx.txt",skip=6)
for (col in colnames(spViaUse)) spViaUse[,col]=as.character(y$Species[spViaUse[,col]])

# demonstrate how to use factors:
spViaLevels <- read.table("spOutrfNN.txt",skip=6)
for (col in colnames(spViaLevels)) spViaLevels[,col]=levels(y$Species)[spViaLevels[,col]]

identical(spViaLevels,spViaUse)

if (require(randomForest))
{
  # build a randomForest predictor
  rf <- randomForest(x=iris[refs,1:4],y=iris[refs,5])
  AsciiGridPredict(rf,xfiles,list(predict="spOutrf.txt"),
  sprf <- t(as.matrix(read.table("spOutrf.txt",skip=6)))
} else sprf <- NULL

# reset the directory to that where the example was started.
setwd(curdir)

par(mfcol=c(2,2),mar=c(1,1,2,1))
image(spOrig,main="Original",col=c("red","green","blue"),
  axes=FALSE,useRaster=TRUE)
image(sprfNN,main="Using Impute",col=c("red","green","blue"),
  axes=FALSE,useRaster=TRUE)
if (!is.null(sprf))
  image(sprf,main="Using Predict",col=c("red","green","blue"),
    axes=FALSE,useRaster=TRUE)
image(dist,main="Neighbor Distances",col=terrain.colors(15),
  axes=FALSE,useRaster=TRUE)
**bestVars**

*Computes the number of best X-variables*

**Description**

The number of best variables is estimated by finding an apparent inflection point in the relationship between the generalized root mean square distance (see `grmsd`) and the number of X-variables.

**Usage**

```r
bestVars(obj, nbest=NULL)
```

**Arguments**

- `obj`: an object created by `varSelection`
- `nbest`: number of variables designated as the best; if null the number is estimated

**Value**

An character vector of variable names in decreasing order of importance.

**Author(s)**

Nicholas L. Crookston <ncrookston.fs@gmail.com>

**See Also**

`varSelection`

**Examples**

```r
require(yaImpute)

data(iris)
set.seed(12345)

x <- iris[,1:2] # Sepal.Length Sepal.Width
y <- iris[,3:4] # Petal.Length Petal.Width

vsel <- varSelection(x=x, y=y, nboot=5, useParallel=FALSE)

bestVars(vsel)
```
Description

Several objects of class `yai` are combined into a new object forming a consensus among the many. The intention is that the many would be formed by running `yai` several times with `bootstrap=TRUE` or by varying other options.

Usage

```r
buildConsensus(reps, noTrgs=FALSE, noRefs=FALSE, k=NULL)
```

Arguments

- **reps**
  - a list of objects class `yai`.
- **noTrgs**
  - If TRUE neighbor relationships for target observations are not merged.
- **noRefs**
  - If TRUE neighbor relationships for reference observations are not merged.
- **k**
  - If not specified, the minimum value of `k` among the objects is used.

Value

An object of class `yai`

Author(s)

Nicholas L. Crookston <ncrookston.fs@gmail.com>
John Coulston <jcoulston@fs.fed.us>

See Also

`yai`

Examples

```r
require (yaImpute)
data(iris)
set.seed(123)

# 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
```
reps <- replicate(20, yai(x=x, y=y, method="msn", bootstrap=TRUE, k=2), simplify=FALSE)
buildConsensus(reps)

---

**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", scale=TRUE)

**Arguments**

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

**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. The scale values (if used) are returned as an attribute (all if some are different than others, a warning is issued).

**Author(s)**

Nicholas L. Crookston <ncrookston.fs@gmail.com>
Andrew O. Finley <finleya@msu.edu>

**See Also**

- `yai.plot.compare.yai`, `impute.yai`, `rmsd.yai`
Examples

```r
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

```r
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 (Stage and Crookston 2007).

Value

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

**Author(s)**

Nicholas L. Crookston <ncrookston.fs@gmail.com>
Andrew O. Finley <finleya@msu.edu>

**References**


**See Also**

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

---

**correctBias**

*Correct bias by selecting different near neighbors*

**Description**

Change the neighbor selections in a `yai` object such that bias (if any) in the average value of an expression of one or more variables is reduced to be within a defined confidence interval.

**Usage**

`correctBias(object, trgVal, trgValCI=NULL, nStdev=1.5, excludeRefIds=NULL, trace=FALSE)`

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>an object of class <code>yai</code> with k &gt; 1.</td>
</tr>
<tr>
<td><code>trgVal</code></td>
<td>an expression defining a variable or combination of variables that is applied to each member of the population (see details). If passed as a character string it is coerced into an expression. The expression can refer to one or more X- and Y-variables defined for the reference observations.</td>
</tr>
<tr>
<td><code>trgValCI</code></td>
<td>The confidence interval that should contain the mean(<code>trgVal</code>). If the mean falls within this interval, the problem is solved. If NULL, the interval is based on <code>nStdev</code>.</td>
</tr>
<tr>
<td><code>nStdev</code></td>
<td>the number of standard deviations in the vector of values used to compute the confidence interval when one is computed, ignored if <code>trgValCI</code> is not NULL.</td>
</tr>
<tr>
<td><code>excludeRefIds</code></td>
<td>identities of reference observations to exclude from the population, if coded as &quot;all&quot; then all references are excluded (see details).</td>
</tr>
<tr>
<td><code>trace</code></td>
<td>if TRUE, detailed output is produced.</td>
</tr>
</tbody>
</table>
Details

Imputation as it is defined in yaimpute can yield biased results. Let's say that you have a collection of reference observations that happen to be selected in a non-biased way among a population. In this discussion, population is a finite set of all individual sample units of interest; the reference plus target observations often represent this population (but this need not be true, see below). If the average of a measured attribute is computed from this random sample, it is an unbiased estimate of the true mean.

Using yai, while setting \( k=1 \), values for each of several attributes are imputed from a single reference observation to a target observation. Once the imputation is done over all the target observations, an average of any one measured attribute can be computed over all the observations in the population. There is no guarantee that this average will be within a pre-specified confidence interval.

Experience shows that despite any lack of guarantee, the results are accurate (not biased). This tends to hold true when the reference data contains samples that cover the variation of the targets, even when they are not a random sample, and even if some of the reference observations are from sample units that are outside the target population.

Because there is no guarantee, and because the reference observations might profitably come from sample units beyond those in the population (so as to insure all kinds of targets have a matching reference), it is necessary to test the imputation results for bias. If bias is found, it would be helpful to do something to correct it.

The correctBias() function is designed to check for, and correct discovered bias by selecting alternative nearby reference observations to be imputed to targets that contribute to the bias. The idea is that even if one reference is closest to a target, its attribute(s) of interest might be greater (or less) than the mean. An alternative neighbor, one that may be almost as close, might reduce the overall bias if it were used instead. If this is the case, correctBias() switches the neighbor selections. It makes as many switches as it can until the mean among the population targets falls within the specified confidence interval. There is no guarantee that the goal will be met.

The details of the method are:

1. An attribute of interest is established by naming one in the call with argument tarVal. Note that this can be a simple variable name enclosed in quotations marks or it can be an expression of one or more variables. If the former, it is converted into an expression that is executed in the environment of the reference observations (both the X- and Y-variables). A confidence interval is computed for this value under the assumption that the reference observations are an unbiased sample of the target population. This may not be the case. Regardless, a confidence interval is necessary and it can alternatively be supplied using trgValCI.

2. One of several possible passes through the data are taken to find neighbor switches that will result in the bias being corrected. A pass includes computing the attribute of interest by applying the expression to values imputed to all the targets, under the assumption that the next neighbor is used in place of the currently used neighbor. This computation results in a vector with one element for each target observation that measures the contribution toward reducing the bias that would be made if a switch were made. The target observations are then ordered into increasing order of how much the distance from the currently selected reference would increase if the switch were to take
place. Enough switches are made in this order to correct the bias. If the bias is not corrected by
the first pass, another pass is done using the next neighbor(s). The number of possible passes is
equal to $k-1$ where $k$ is set in the original call to \texttt{yai}. Note that switches are made among targets
only, and never among reference observations that may make up the population. That is, reference
observations are always left to represent themselves with $k=1$.

3. Here are details of the argument \texttt{excludeRefIds}. When computing the mean of the attribute
of interest (using the expression), \texttt{correctBias()} must know which observations represent the
population. Normally, all the target observations would be in this set, but perhaps not all of the
reference observations. When \texttt{excludeRefIds} is left NULL, the population is made of all reference
and all target observations. Reference observations that should be left out of the calculations
because they are not part of the population can be specified using the \texttt{excludeRefIds} argument
as a vector of character strings identifying the rownames to leave out, or a vector of row numbers
that identify the row numbers to leave out. If \texttt{excludeRefIds="all"}, all reference observations are
excluded.

\textbf{Value}

An object of class \texttt{yai} where $k=1$ and the neighbor selections have been changed as described
above. In addition, the call element is changed to show both the original call to \texttt{yai} and the call to
this function. A new list called \texttt{biasParameters} is added to the \texttt{yai} object with these tags:

\begin{itemize}
  \item \texttt{trgValCI} the target CI.
  \item \texttt{curVal} the value of the bias that was achieved.
  \item \texttt{npasses} the number of passes through the data taken to achieve the result.
  \item \texttt{oldk} the old value of $k$.
\end{itemize}

\textbf{Author(s)}

Nicholas L. Crookston <ncrookston.fs@gmail.com>

\textbf{See Also}

\texttt{yai}

\textbf{Examples}

data(iris)
set.seed(12345)

# 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.
spl <- as.integer(iris$Species=="setosa")
ensembleImpute <- data.frame(cbind(iris[,1],sp1,sp2),row.names=rownames(iris))
y2 <- y2[refs,]
names(y2) <- c("Petal.Width","Sp1","Sp2")

# find 5 reference neighbors for each target
msn <- yai(x=x,y=y2,method="msn",k=5)

# check for and correct for bias in mean "Petal.Width". Neighbor
# selections will be changed as needed to bring the imputed values
# into line with the CI. In this case, no changes are made (npasses
# returns as zero).
msnCorr = correctBias(msn,trgVal="Petal.Width")
msnCorr$biasParameters

---

ensembleImpute  Computes the mean, median, or mode among a list of impute.yai objects

Description
Several objects of class impute.yai or yai are combined by computing the mean, median, or mode
of separate, individual imputations. The intention is that the members of the first argument would
be formed by running yai several times with bootstrap=TRUE or by varying other options.

Usage
ensembleImpute(imputes, method="mean",...)

Arguments
imputes  a list of objects class impute.yai or yai. Function impute.yai is called for list
members where the class is yai.

method  when "mean", the continuous variables are averaged using mean, otherwise the
median is used. Mode is always used for character data (generally the case for
factors).

...  passed to impute.yai.

Value
An object of class c("impute.yai","data.frame"), see impute.yai. The attributes of the data.frame
include the following:

- sd  A data.frame of standard deviations for continuous variables if there are any. The columns
are not reported if the standard deviation is zero for all observations which is typically true of
"observed" values.
errorStats

• \(N\) the number of replications used to compute the corresponding data; reported only if the
number differs from the total number of replications. This will be the case when bootstrap,
sampleVar, or both are used in \textit{yai}.

• methods the method used for each variable.

Author(s)

Nicholas L. Crookston <ncrookston.fs@gmail.com>
John Coulston <jcoulston@fs.fed.us>

See Also

\textit{yai} \textit{buildConsensus} \textit{impute.yai}

Examples

```r
require (yaImpute)
data(iris)
set.seed(123)

# 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

reps <- replicate(10, yai(x=x,y=y,method="msn",bootstrap=TRUE,k=2),
simplify=FALSE)

ensembleImpute(reps,ancillaryData=iris)
```

---

\textbf{errorStats} \quad \textit{Compute error components of k-NN imputations}

Description

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

Usage

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

mahal

An object of class yai computed with method="mahalanobis".

... 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 reference observations that are zero distance from each other (used to compute rmmsd0).

plg

The upper tail p-value used to pick reference observations that are substantially distant from each other (used to compute rmstdlg).

seeMethod

Method used to compute SEE: seeMethod="lm" uses lm and seeMethod="gam" uses gam. In both cases, the model formula is a simple linear combination of the X-variables.

Details

See https://academic.oup.com/forestscience/article/53/1/62/4604364

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 the 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 yai object.

names of ... arguments error statistics for each of the remaining yai 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 method="mahalanobis" (always based on the first argument to the function).

mlf square root of the model lack of fit: sqrt(see^2 − (rmmsd0^2/2)).

rmstd root mean square error.

rmstdlg root mean square error of the observations with larger distances.

sei standard error of imputation sqrt(rmstd^2 − (rmmsd0^2/2)).

dstc distance component: sqrt(rmstd^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@gmail.com>

Albert R. Stage
References


See Also

`yai`, `TallyLake`

Examples

```r
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],
    noTrgs=TRUE,method="mahalanobis")

msn=yai(x=TallyLake[,9:29],y=TallyLake[,1:8],
    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

```r
foruse(object,kth=NULL,method="kth",targetsOnly=FALSE)
```
Arguments

- **object**: an object created by `yai`
- **kth**: when NULL (and method="kth"), the best pick is reported (a reference observation represents itself), otherwise the kth neighbor is picked.
- **method**: the method used to select references to represent observations, as follows:
  - **kth**: the kth nearest neighbor is picked;
  - **random**: for each observation, the value of kth is selected at random from the 1 to k neighbors (1 to kth if kth specified);
  - **randomWeighted**: 1/(1+d) is used as a probability weight factor in selecting the value of kth, where d is the distance..
- **targetsOnly**: when is TRUE, reporting of references is not done.

Author(s)

Nicholas L. Crookston <ncrookston.fs@gmail.com>
Andrew O. Finley <finleya@msu.edu>

Examples

```r
require(yaImpute)
data(iris)

# form some test data
set.seed(1234)
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))
```

---

**grmsd**

*Generalized Root Mean Square Distance Between Observed and Imputed Values*
Description

Computes the root mean square distance between predicted and corresponding observed values in an orthogonal multivariate space. This value is the mean Mahalanobis distance between observed and imputed values in a space defined by observations and variables were observed and predicted values are defined. The statistic provides a way to compare imputation (or prediction) results. While it is designed to work with imputation, the function can be used with objects that inherit from `lm` or with matrices and data frames that follow the column naming convention described in the details.

Usage

```r
grmsd(..., ancillaryData=NULL, vars=NULL, wts=NULL, rtnVectors=FALSE, imputeMethod="closest")
```

Arguments

- `...`: objects created by any combination of `yai`, `impute.yai`, `ensembleImpute`, `buildConsensus`, `lm` and data frames or matrices that follow the column naming convention described in the details below. If an object is of class `yai`, a call to `impute.yai` is generated internally.
- `ancillaryData`: a data frame that defines variables, passed to `impute.yai`.
- `vars`: a list of variable names you want to include; if NULL all available variables are included (note that if codeimpute.yai the Y-variables are returned when `vars=NULL`).
- `wts`: A vector of weights used to compute the mean distances, see details below.
- `rtnVectors`: The vectors of individual distances are returned (see Value) rather than the mean Mahalanobis distance.
- `imputeMethod`: passed as method to `impute.yai`.

Details

This function is designed to compute the root mean square distance between observed and predicted observations over several variables at once. It is the Mahalanobis distance between observed and predicted but the name emphasizes the similarities to root mean square difference (or error, see `rmsd`). Here are some notable characteristics.

1. In the univariate case this function returns the same value as `rmsd` with `scale=TRUE`. In that case the root mean square difference is computed after `scale` has been called on the variable.
2. Like `rmsd`, `grmsd` is zero if the imputed values are exactly the same as the observed values over all variables.
3. Like `rmsd`, `grmsd` is approximately 1.0 when the mean of each variable is imputed in place of a near neighbor (it would be exactly 1.0 if the maximum likelihood estimate of the covariance were used rather than the unbiased estimate – it approaches 1 as n gets large.) This situation corresponds to regression where the slope is zero. It indicates that the imputation error is, over all, the same as if the means of the variables were imputed rather than near neighbors (it does not signal that the means were imputed).
4. Like `rmsd`, values of `grmsd` > 1.0 indicate that, on average, the errors in the imputation are greater than they would be if the mean of the corresponding variables were imputed for each observation.
5. Note that individual \textit{rmsd} values can be computed even when the variance of the variable is zero. In contrast, \textit{grmsd} can only be computed in the situation where the observed data matrix is full rank. Rank is determined using \texttt{qr} and columns are removed from the analysis to create this condition if necessary (with a warning).

Observed and predicted are matched using the column names. Column names that have "\_o" are matched to those that do not. Columns that do not have matching observed and imputed (predicted) values are ignored.

Several objects may be passed as "...". Function \texttt{impute.yai} is called for any objects that were created by \texttt{yai}; ancillaryData and \texttt{vars} are passed to \texttt{impute.yai} when it is used.

When objects inherit from \texttt{lm}, a suitable matrix is formed using by calling the \texttt{predict} and \texttt{resid} functions.

Factors, if found, are removed (with a warning).

When argument \texttt{wts} is defined there must be one value for each pair of observed and predicted variables. If the values are named (preferred), then the names are matched to the names of predicted variables (no \_o suffix). The matched values effectively scale the axes in which distances are computed. When this is done, the resulting distances are not Mahalanobis distances.

\textbf{Value}

When \texttt{rtnVectors=FALSE}, a sorted named vector of mean distances is returned; the names are taken from the arguments.

When \texttt{rtnVectors=TRUE} the function returns vectors of distances, sorted and named as done when this argument is \texttt{FALSE}.

\textbf{Author(s)}

Nicholas L. Crookston <ncrookston.fs@gmail.com>

\textbf{See Also}

\texttt{yai, impute.yai, rmsd.yai, notablyDifferent}

\textbf{Examples}

```
require(yaImpute)

data(iris)
set.seed(12345)

# 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")
```
# compute the average distances between observed and imputed (predicted)
grmsd(msn, mal, lmFit=lm(as.matrix(y) ~ ., data=x[refs,]))

# use the all variables and observations in iris
# Species is a factor and is automatically deleted with a warning
grmsd(msn, mal, ancillaryData=iris)

# here is an example using lm, and another using column
# means as predictions.

impMean <- y
colnames(impMean) <- paste0(colnames(impMean), "o")
impMean <- cbind(impMean, y)
# set the predictions to the mean's of the variables
impMean[, "Petal.Length"] <- mean(impMean[, "Petal.Length"])
impMean[, "Petal.Width"] <- mean(impMean[, "Petal.Width"])

grmsd(msn, mal, lmFit=lm(as.matrix(y) ~ ., data=x[refs,]), impMean)

# compare to using function rmsd (values match):
msnimp <- na.omit(impute(msn))
grmsd(msnimp[, c("Petal.Length","Petal.Length.o")])
rmsd(msnimp[, c("Petal.Length","Petal.Length.o")], scale=TRUE)

# these are multivariate cases and they don't match
# because the covariance of the two variables is > 0.
grmsd(msnimp)
colSums(rmsd(msnimp, scale=TRUE))/2

# get the vectors and make a boxplot, identify outliers
stats <- boxplot(grmsd(msn, mal, ancillaryData=iris[,-5], rtnVectors=TRUE),
  ylab="Mahalanobis distance")
stats$out
# 118 132
#2.231373 1.990961

---

**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**

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

object an object of class yai.

ancillaryData 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.

method the method used to compute the imputed values for continuous variables, as follows:
closest: use the single neighbor that is closest (this is the default and is always used when k=1);
mean: the mean of the k neighbors is taken;
median: the median of the k neighbors is taken;
dstWeighted: a weighted mean 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 or median: actually is the mode—it is the factor level that occurs the most often among the k neighbors;
dstWeighted: a mode 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 behavior depends on the value of ancillaryData: when it is NULL, the Y-variables are imputed and otherwise all present in ancillaryData are imputed.

observed when TRUE, columns are created for observed values (those from the target observations) as well as imputed values (those from the reference observations).

... passed to other methods, currently not used.

Value

An object of class c("impute.yai" ,"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@gmail.com>
Andrew O. Finley <finleya@msu.edu>
Emilie Henderson <emilie.henderson@oregonstate.edu>
See Also
yai

Examples

```r
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 LiDAR 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:

- **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


---

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

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

Author(s)

Nicholas L. Crookston <ncrookston.fs@gmail.com>
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")
newtargets

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**

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

**Arguments**

- **object**: an object of class `yai`.
- **newdata**: a data frame or matrix of new targets for which neighbors are are found. Must include at least the X-variables used in the original call to `yai`.
- **k**: if NULL, the value is taken from `object`, otherwise the number of nearest neighbors to find.
- **ann**: if NULL, the value is taken from `object`. When TRUE `ann` is used to find neighbors, and when FALSE a slow exact search is used.

**Value**

An object of class `yai`, that 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 matrix of distances between a target (identified by its row name) and the k references. There are k columns.
- **neiIdsTrgs**: a matrix of reference identifications that correspond to neiDstTrgs.
- **neiDstRefs**: set NULL as if noRefs=TRUE in the original call to `yai`.
- **neiIdsRefs**: set NULL as if noRefs=TRUE in the original call to `yai`.
- **k**: the value of k, replaced if changed.
- **ann**: the value of the ann argument.
Author(s)
Nicholas L. Crookston <ncrookston.fs@gmail.com>
Andrew O. Finley <finleya@msu.edu>

See Also
yai

Examples

require (yaImpute)

data(iris)

# set the random number seed so that example results are consistant
# 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))

if (require(randomForest))
{
  # 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)),])
}
notablyDifferent

Finds observations with large differences between observed and imputed values

Description

This routine identifies observations with large errors as measured by scaled root mean square error (see `rmsd.yai`). A threshold is used to detect observations with large differences.

Usage

```r
notablyDifferent(object, vars=NULL, threshold=NULL, p=.05, ...)
```

Arguments

- **object**: an object of class `yai`.
- **vars**: a vector of character strings naming the variables to use, if null the X-variables form `object` are used.
- **threshold**: a threshold that if exceeded the observations are listed as notably different.
- **p**: 
  
  \[(1-p)*100\] is the percentile point in the distribution of differences used to compute the threshold (used when `threshold` is `NULL`).
- **...**: additional arguments passed to `impute.yai`.

Details

The scaled differences are computed a follows:

1. A matrix of differences between observed and imputed values is computed for each observation (rows) and each variable (columns).
2. These differences are scaled by dividing by the standard deviation of the observed values among the reference observations.
3. The scaled differences are squared.
4. Row means are computed resulting in one value for each observation.
5. The square root of each of these values is taken.

These values are Euclidean distances between the target observations and their nearest references as measured using specified variables. All the variables that are used must have observed and imputed values. Generally, this will be the X-variables and not the Y-variables.

When `threshold` is `NULL`, the function computes one using the `quantile` function with its default arguments and `probs=1-p`.
notablyDifferent

Value
A named list of several items. In all cases vectors are named using the observation ids which are the row names of the data used to build the `yai` object.

call
The call.
vars
The variables used (may be fewer than requested).
threshold
The threshold value.

notablyDifferent.refs
A sorted named vector of references that exceed the threshold.

notablyDifferent.trgs
A sorted named vector of targets that exceed the threshold.

rmsdS.refs
A sorted named vector of scaled RMSD references.

rmsdS.trgs
A sorted named vector of scaled RMSD targets.

Author(s)
Nicholas L. Crookston <ncrookston.fs@gmail.com>

See Also

notablyDistant, plot.notablyDifferent, yai, grmsd

Examples

data(iris)

set.seed(12345)

# 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")

notablyDifferent(msn)
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,method="distribution")

Arguments

object  
an object of class yai.

kth  
the kth neighbor is used.

threshold  
the threshold distance that identifies notably large distances between observations.

p  
(1-p)*100 is the percentile point in the distribution of distances used to compute the threshold (only used when threshold is NULL).

method  
the method used to compute the threshold, see details.

Details

When threshold is NULL, the function computes one using one of two methods. When method is "distribution", assumption is made that 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, which is the point in the distribution where a fraction, p, of the neighbors are larger than the threshold.

When method is "quantile", the function uses the quantile function with probs=1-p.

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, 3) the threshold used, and 4) the method used.

Author(s)

Nicholas L. Crookston <ncrookston.fs@gmail.com>

See Also

notablyDifferent yai
Examples

data(iris)

set.seed(12345)

# 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

```r
## 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@gmail.com>
Andrew O. Finley <finleya@msu.edu>
See Also

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

plot.notablyDifferent  Plots the scaled root mean square differences between observed and predicted

Description

Provides a descriptive plot of the Imputation Error Profile for object(s) created by notablyDifferent.

Usage

## S3 method for class 'notablyDifferent'
plot(x, add=FALSE, ...)

Arguments

x 1. an object create by notablyDifferent, or 2. a (named) list of such objects.
add set TRUE if you want to add this plot to an existing plot.
... passed to plot functions.

Author(s)

Nicholas L. Crookston <ncrookston.fs@gmail.com>

See Also

notablyDistant and yai

Examples

require(yaImpute)
data(iris)
set.seed(12345)

# 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 <- notablyDifferent(yai(x=x, y=y, method="mahalanobis"), vars=colnames(x))
if (require(randomForest)) {
  
}
plot.varSel

Boxplot of mean Mahalanobis distances from varSelection()

Description

Provides a descriptive plot of how the mean Mahalanobis distances change as variables are added or deleted using varSelection.

Usage

## S3 method for class 'varSel'
plot(x, main=NULL, nbest=NULL, arrows=TRUE, ...)

Arguments

x an object created by varSelection
main becomes the plot title, if NULL one is generated
nbest number of variables designated in the plot as the best; if NULL the number is computed by bestVars
arrows if true, an arrow is added to the plot designating the best variables.
... passed to boxplot functions

Author(s)

Nicholas L. Crookston <ncrookston.fs@gmail.com>

See Also

varSelection and yai

Examples

require(yaImpute)
data(iris)
set.seed(12345)
x <- iris[,1:2]  # Sepal.Length Sepal.Width
y <- iris[,3:4]  # Petal.Length Petal.Width
vsel <- varSelection(x=x, y=y, nboot=5, useParallel=FALSE)
plot(vsel)

rf <- notablyDifferent(yai(x=x, y=y, method="randomForest"), vars=colnames(x))
plot.notablyDifferent(list(Mahalanobis=mal, randomForest=rf))
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 impute.yai, or
2. an object created by yai.

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@gmail.com>
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)
Generic predict function for class \texttt{yai}

Description

Provides a generic interface for getting predicted values for \texttt{yai} objects.

Usage

\begin{verbatim}
## S3 method for class 'yai'
predict(object, newdata, \ldots)
\end{verbatim}

Arguments

- \texttt{object}: an object of class \texttt{yai} which is passed as argument to \texttt{newtargets}, \texttt{impute.yai}, or both of these functions, see details.
- \texttt{newdata}: a data frame that at a minimum contains the \texttt{X}-variables for new observations.
- \texttt{\ldots}: passed to \texttt{newtargets} and \texttt{impute.yai}, see details.

Details

When argument \texttt{newdata} is present function \texttt{newtargets} is called followed by a call to \texttt{impute.yai}. If include in the \ldots, the arguments \texttt{k} and \texttt{ann} are passed to \texttt{newtargets}.

When argument \texttt{newdata} is absent, \texttt{impute.yai} is called without first calling \texttt{newtargets}.

All of the \ldots arguments are passed to \texttt{impute.yai}.

Another form of prediction in imputation is to get the identity of the imputed observations. Use function \texttt{foruse} for this purpose.

Value

An object of class \texttt{impute.yai}.

Author(s)

Nicholas L. Crookston <ncrookston.fs@gmail.com>

See Also

\texttt{foruse}, \texttt{newtargets} \texttt{impute.yai}
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@gmail.com>
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. The RMSD values can be scaled to afford comparisons among variables.

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), if a named vector, the values are scaled by the corresponding values.
- **...**: 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`, or set of values is supplied, RMSD is divided by the scaling factor. The scaling factor is the standard deviation of the reference observations under the assumption that they are representative of the population.

Value

A data frame with the row names as vars and the column as `rmsd`. When `scale=TRUE`, the column name is `rmsdS`. The scaling factors used, if any, are returned as an attribute.

Author(s)

Nicholas L. Crookston <ncrookston.fs@gmail.com>
Andrew O. Finley <finleya@msu.edu>

See Also


### TallyLake

<table>
<thead>
<tr>
<th>TallyLake</th>
<th>Tally Lake, Flathead National Forest, Montana, USA</th>
</tr>
</thead>
</table>

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 (polygons).

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
• eevsqrdd^2\) Mean of 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
• insom 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

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@gmail.com>
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
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@gmail.com>
Andrew O. Finley <finleya@msu.edu>

See Also

yai
Description

Computes \texttt{grmsd} (generalized root mean square distance) as variables are added to (\texttt{method="addVars"}) or removed from (\texttt{method="delVars"}) an k-NN imputation model. When adding variables the function keeps variables that strengthen imputation and deletes that weaken the imputation the least. The measure of model strength is \texttt{grmsd} between imputed and observed \texttt{Y}-variables among the reference observations.

Usage

\begin{verbatim}
varSelection(x,y,method="addVars",yaiMethod="msn",imputeMethod="closest",
wts=NULL,nboot=20,trace=FALSE,
useParallel=if (.Platform$OS.type == "windows") FALSE else TRUE,...)
\end{verbatim}

Arguments

- \texttt{x} a set of \texttt{X}-Variables as used in \texttt{yai}.
- \texttt{y} a set of \texttt{Y}-Variables as used in \texttt{yai}.
- \texttt{method} if \texttt{addVars}, the \texttt{X}-Variables are added and if \texttt{delVars} they are deleted (see details).
- \texttt{yaiMethod} passed as \texttt{method} to \texttt{yai}.
- \texttt{imputeMethod} passed as \texttt{method} to \texttt{impute.yai}.
- \texttt{wts} passed as argument \texttt{wts} to \texttt{grmsd} which is used to score the alternative variable sets.
- \texttt{nboot} the number of bootstrap samples used at each variable selection step (see Details). When \texttt{nboot} is zero, NO bootstrapping is done.
- \texttt{trace} if \texttt{TRUE} information at each step is output.
- \texttt{useParallel} function \texttt{link{parallel:mclapply}} from \texttt{parallel} will be used if it is available for running the bootstraps. It it is not available, \texttt{link{lapply}} is used (which is the only option on windows).
- \texttt{...} passed to \texttt{link{yai}}

Details

This function tracks the effect on generalized root mean square distance (see \texttt{grmsd}) when variables are added or deleted one at a time. When adding variables, the function starts with none, and keeps the single variable that provides the smallest \texttt{grmsd}. When deleting variables, the functions starts with all \texttt{X}-Variables and deletes them one at a time such that those that remain provide the smallest \texttt{grmsd}. The function uses the following steps:
1. Function `yai` is run for all the Y-variables and candidate X-variable(s). The result is passed to `impute.yai` to get imputed values of Y-variables. That result is passed to `grmsd` to compute a mean Mahalanobis distance for the case where the candidate variable is included (or deleted depending on `method`). However, these steps are done once for each bootstrap replication and the resulting values are averaged to provide an average mean Mahalanobis distance over the bootstraps.

2. Step one is done for each candidate X-variable forming a vector of `grmsd` values, one corresponding to the case where each candidate is added or deleted.

3. When variables are being added (method=“addVars”), the variable that is related to the smallest `grmsd` is kept. When variables are being deleted (method=“delVars”), the variable that is related to the largest `grmsd` is deleted.

4. Once a variable has been added or deleted, the function proceeds to select another variable for selection or deletion by considering all remaining variables.

**Value**

An list of class `varSel` with these tags:

- `call` the call
- `grmsd` a 2-column matrix of the mean and std dev of the mean Mahalanobis distances associated with adding or removing the variables stored as the rownames. When `nboot<2`, the std dev are NA
- `allgrmsd` a list of the `grmsd` values that correspond to each bootstrap replication. The data in `grmsd` are based on these vectors of information.
- `method` the value of argument `method`.

**Author(s)**

Nicholas L. Crookston <ncrookston.fs@gmail.com>

**See Also**

`yai`, `impute.yai`, `bestVars` and `grmsd`

**Examples**

```r
data(iris)
set.seed(12345)
x <- iris[,1:2] # Sepal.Length Sepal.Width
y <- iris[,3:4] # Petal.Length Petal.Width
vsel <- varSelection(x=x,y=y,nboot=5,useParallel=FALSE)
vsel
bestVars(vsel)
```
whatsMax

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 \( n_{big} \) 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

```r
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@gmail.com>
Andrew O. Finley <finleya@msu.edu>

Examples

```r
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])
```
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

```r
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", bootstrap=FALSE, ppControl=NULL, sampleVars=NULL, 
rfXsubsets=NULL)
```

Arguments

- **x**: 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 x, one must be used for y as well, if needed.

- **y**: 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.

- **data**: when x and y are formulas, then data 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 yai into two sets.

- **k**: the number of nearest neighbors; default is 1.

- **noTrgs**: when TRUE, skip finding neighbors for target observations.

- **noRefs**: when TRUE, skip finding neighbors for reference observations.

- **nVec**: number of canonical vectors to use (methods msn and msn2), or number of independent of X-variables reference data when method mahaLarobis. When NULL, the number is set by the function.

- **pVal**: significant level for canonical vectors, used when method is msn or msn2.

- **method**: is the strategy used for computing distance and therefore for finding neighbors; the options are quoted key words (see details):


- euclidean distance is computed in a normalized X space.
- raw like euclidean, except no normalization is done.
- mahalanobis distance is computed in its namesakes space.
- ica like mahalanobis, but based on Independent Component Analysis using package fastICA.
- msn distance is computed in a projected canonical space.
- msn2 like msn, but with variance weighting (canonical regression rather than correlation).
- msnPP like msn, except that the canonical correlation is computed using projection pursuit from ccaPP (see argument ppControl).
- gower distance is computed in its namesakes space using function gower_topn from package gower; forces ann to be FALSE.

<table>
<thead>
<tr>
<th>ann</th>
<th>TRUE if ann is used to find neighbors, FALSE if a slow search is used.</th>
</tr>
</thead>
<tbody>
<tr>
<td>mtry</td>
<td>the number of X-variables picked at random when method is randomForest, see randomForest, default is sqrt(number of X-variables).</td>
</tr>
<tr>
<td>ntree</td>
<td>the number of classification and regression trees when method is randomForest. When more than one Y-variable is used, the trees are divided among the variables. Alternatively, ntree can be a vector of values corresponding to each Y-variable.</td>
</tr>
<tr>
<td>rfMode</td>
<td>when buildClasses and method is randomForest, continuous variables are internally converted to classes forcing randomForest to build classification trees for the variable. Otherwise, regression trees are built if your version of randomForest is newer than 4.5-18.</td>
</tr>
<tr>
<td>bootstrap</td>
<td>if TRUE, the reference observations are sampled with replacement.</td>
</tr>
<tr>
<td>ppControl</td>
<td>used to control how canonical correlation analysis via projection pursuit is done, see Details.</td>
</tr>
<tr>
<td>sampleVars</td>
<td>the X- and/or Y-variables will be sampled (without replacement) if this is not NULL and greater than zero. If specified as a single unnamed value, that value is used to control the sample size of both X and Y variables. If two unnamed values, then the first is taken for X-variables and the second for Y-variables. If zero, no sampling is done. Otherwise, values are less than 1.0 they are taken as the proportion of the number of variables. Values greater or equal to 1 are number of variables to be included in the sample. Specification of a large number will cause the sequence of variables to be randomized.</td>
</tr>
<tr>
<td>rfXsubsets</td>
<td>a named list of character vectors where there is one vector for each Y-variable, see details, only applies when method=&quot;randomForest&quot;</td>
</tr>
</tbody>
</table>
Details

See the paper at doi:10.18637/jss.v023.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, random, 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.

When bootstrap=TRUE the reference observations are sampled with replacement. The sample size is set to the number of reference observations. Normally, about a third of the reference observations are left out of the sample; they are often called out-of-bag samples. The out-of-bag observations are then treated as targets.

When method="msnPP" projection pursuit from ccaPP is used. The method is further controlled using argument ppControl to specify a character vector that has two named components.

- method One of the following "spearman", "kendall", "quadrant", "M", "pearson", default is "spearman"
- search If "data" or "proj", then ccaProj is used, otherwise the default ccaGrid is used.

Here are some details on argument rfXsubsets. When method="randomForest" one call to randomForest is generated for each Y-variable. When argument rfXsubsets is left NULL, all the X-variables are used for each of the Y-variables. However, sometimes better results can be achieved by using specific subsets of X-variables for each Y-variable. This is done by setting rfXsubsets equal to a named list of character vectors. The names correspond to the Y-variable names and the character vectors hold the list of X-variables for the corresponding Y-variable.

Value

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

- call the call.
- yRefs, xRefs matrices of the X- and Y-variables for just the reference observations (unscaled). The scale factors are attached as attributes.
- 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.
- cancort an object of class cca (from package vegan) when method msn or msn2 is used (NULL otherwise).
- ccaVegan an object of class cca (from package vegan) when method gmn is used.
- ftest a list containing partial F statistics and a vector of Pr>F (pgf) corresponding to the canonical correlation coefficients when method msn or msn2 is used (NULL otherwise).
- yScale, xScale scale data used on yRefs and xRefs as needed.
the value of $k$.

pVal as input; only used when method msn, msn2 or msnPP is used.

projector NULL when not used. For methods msn, msn2, msnPP, gnn and mahalanobis, this is a matrix that projects normalized X-variables into a space suitable for doing Eculidian distances.

nVec number of canonical vectors used (methods msn and msn2), or number of independent X-variables in the reference data when method mahalanobis is used.

method as input, the method used.

ranForest a list of the forests if method randomForest is used. There is one forest for each Y-variable, or just one forest when there are no Y-variables.

ICA a list of information from fastICA when method ica is used.

ann the value of ann, TRUE when ann 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 matrix of distances between a target (identified by its row name) and the $k$ references. There are $k$ columns.

neiIdsTrgs a matrix of reference identifications that correspond to neiDstTrgs.

neiDstRefs, neiIdsRefs counterparts for references.

bootstrap a vector of reference rownames that constitute the bootstrap sample; or the value FALSE when bootstrap is not used.

Author(s)
Nicholas L. Crookston <ncrookston.fs@gmail.com>
John Coulston <jcoulston@fs.usda.gov>
Andrew O. Finley <finleya@msu.edu>

See Also
grmsd ensembleImpute

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
```r
# Petal.Length Petal.Width

y <- iris[refs,3:4]

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

# compare these results using the generalized mean distances. mal wins!
grmsd(mal,msn)

# use projection pursuit and specify ppControl (loads package ccaPP)
if (require(ccaPP)) {
  msnPP <- yai(x=x,y=y,method="msnPP",ppControl=c(method="kendall",search="proj"))
  grmsd(mal,msnPP,msn)
}

#########
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]

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

# the results can be plotted.
plot(mal,vars=yvars(mal)[1:16])

# compare these results using the generalized mean distances..
grmsd(mal,msn)

# try method="gower"
if (require(gower)) {
  gow <- yai(x=x, y=y, method="gower", k=1)
  # compare these results using the generalized mean distances..
grmsd(mal,msn,gow)
}

# try method="randomForest"
if (require(randomForest)) {
  # 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 using individual rmsd's
compare.yai(mal,msn,rforig)
plot(compare.yai(mal,msn,rforig))

# build another randomForest case forcing regression
# to be used for continuous variables. The answers differ
# but one is 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(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

object an object of class yai.
nTop the nTop 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.
**yaiVarImp**

**Author(s)**

Nicholas L. Crookston <ncrookston.fs@gmail.com>
Andrew O. Finley <finleya@msu.edu>

**See Also**

*yai*, *yaiVarImp*

---

**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 using `scale`. By default the scores are plotted and scores themselves are invisibly returned. For classification, the scores are derived from "MeanDecreaseAccuracy" and for regression they are based in " using `importance`.

**Usage**

```r
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 the `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@gmail.com>

**See Also**

*yai, yaiRFsummary, compare.yai*
Examples

if (require(randomForest))
{
  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

* datasets
  MoscowMtStJoe, 29
  TallyLake, 44
* hplot
  plot.compare.yai, 38
  plot.notablyDifferent, 39
  plot.yai, 41
* misc
  ann, 2
  cor.yai, 16
  correctBias, 17
  impute.yai, 27
  mostused, 32
  notablyDistant, 37
  unionDataJoin, 46
  vars, 47
  whatsMax, 50
  yaiRFsummary, 56
  yaiVarImp, 57
* multivariate
  applyMask, 6
  bestVars, 13
  buildConsensus, 14
  compare.yai, 15
  cor.yai, 16
  correctBias, 17
  ensembleImpute, 20
  errorStats, 21
  foruse, 23
  grmsd, 24
  impute.yai, 27
  mostused, 32
  newtargets, 33
  notablyDifferent, 35
  notablyDistant, 37
  plot.varSel, 40
  rmsd.yai, 43
  varSelection, 48
  yai, 51
  yaiRFsummary, 56
  yaiVarImp, 57
* predict
  predict.yai, 42
* print
  print.yai, 43
* spatial
  AsciiGridImpute, 8
* tree
  yaiRFsummary, 56
  yaiVarImp, 57
* utilities
  AsciiGridImpute, 8
  ann, 2, 9, 33, 52, 54
  applyMask, 6
  AsciiGridImpute, 8, 33
  AsciiGridPredict (AsciiGridImpute), 8
  attributes, 28
  bestVars, 13, 40, 49
  boxplot, 57
  buildConsensus, 14, 21, 25
  cca, 52
  ccaGrid, 53
  ccaProj, 53
  compare.yai, 15, 38, 39, 57
  cor.yai, 15, 16, 16
  correctBias, 17
  ensembleImpute, 20, 25, 54
  errorStats, 21
  expression, 17, 18
  fastICA, 52, 54
  foruse, 23, 32, 42
  gam, 22
  gower_topn, 52
  grmsd, 13, 24, 36, 48, 49, 54
importance, 57
impute, 10
impute (impute.yai), 27
impute.yai, 15–17, 20, 21, 25, 26, 27, 35, 39, 41–44, 48, 49
invisible, 10

list, 8
lm, 10, 22, 25, 26, 54

MoscowMtStJoe, 29
mostused, 32

newtargets, 6, 7, 10, 33, 42, 53
notablyDifferent, 26, 35, 37, 39
notablyDistant, 36, 37, 39

plot.compare.yai, 15, 38
plot.impute.yai (plot.yai), 41
plot.notablyDifferent, 36, 39
plot.varSel, 40
plot.yai, 41
predict, 8, 10, 26
predict.yai, 42
print.yai, 43

qr, 26
quantile, 35, 37

randomForest, 51–53, 56, 57
rda, 52
resid, 26
rmsd, 25, 26
rmsd (rmsd.yai), 43
rmsd.yai, 15, 17, 26, 35, 39, 43
runif, 52

scale, 25, 57
summary.yai (print.yai), 43

TallyLake, 23, 44
unionDataJoin, 46

vars, 47
varSelection, 13, 40, 48

whatsMax, 50
xvars (vars), 47

yai, 6–10, 14–29, 32–37, 39–44, 47–50, 51, 56, 57
yaImpute (yai), 51
yaiRFsummary, 56, 57
yaiVarImp, 56, 57, 57
yvars (vars), 47