Package ‘yaImpute’

March 31, 2012

Version 1.0-18

Date March 2012

Title yaImpute: An R Package for k-NN Imputation

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

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

Depends R (>= 2.10)

Suggests vegan, randomForest, gam, fastICA

Description Performs popular nearest neighbor routines for imputation

License GPL (>= 2)

Repository CRAN

Date/Publication 2012-03-31 05:20:16

Repository/R-Forge/Project yaimpute

Repository/R-Forge/Revision 17

R topics documented:

---

ann *Approximate nearest neighbor search routines*

Description

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

Usage

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

Arguments

ref	an $n \times d$ matrix containing the reference point set S. Each row in 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+eps) from true i^{th} nearest neighbor, where eps≥ 0 . Specifically, the true (not squared) difference between the true i^{th} and the approximation of the i^{th} point is a factor of (1+eps). The default value of eps=0 is an exact search.
tree.type	the data structures kd-tree or bd-tree as quoted key words *kd* and *bd*, respectively. A brute force search can be specified with the quoted key word *brute*. If *brute* is specified, then all subsequent arguments are ignored. The default is the kd-tree.

`search.type` either standard or priority search in the kd-tree or bd-tree, specified by quoted key words *standard* and *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 kd-tree and bd-tree. Splitting rule options are the quoted key words:

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

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

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

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

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

`verbose` if true, search progress is printed to the screen.

`...` currently no additional arguments.

Details

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

Value

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

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

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

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

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

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

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

bucket.size as defined in the ann function call.

split.rule as defined in the ann function call.

shrink.rule as defined in the ann function call.

Author(s)

Andrew O. Finley <finleya@msu.edu>

Examples

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

m <- 10000

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

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

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

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

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


## Take a reference sample.
```

```
n <- 2000
ref <- c.all[sample(1:nrow(c.all), n),]

## Compare search times
k <- 10
## Do a simple brute force search.
brute <- ann(ref=ref[,1:2], target=c.all[,1:2],
             tree.type="brute", k=k, verbose=FALSE)
print(brute$searchTime)

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

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

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

AsciiGridImpute *Imputes/Predicts data for Ascii Grid maps*

Description

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

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

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

Usage

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

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

Arguments

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

`xfiles` A `list` of input file names where there is one grid file for each X-variable. List elements must be given the same names as the X-variables they correspond with and there must be one file for each X-variable used when `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 `predict` function related to `object` or returned by `myPredFunc`. This form only applies to `AsciiGridPredict`, when the object is not class `yai`.
- (2) A `list` of output file names where there is one grid file for each *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 `yai`: For `distance=`*"filename"* a map of the distances is output and if `useid=`*"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` 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` 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 to predict output using the `object` and newdata from the `xfiles`. Two arguments are passed to this function, the first is the value of `object` and the second is a data frame of the new predictor variables created for each row of data from your input maps. If NULL, the generic `predict` function is called for `object`.

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

Details

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

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

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

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

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

See Also

yai, impute, and newtargets

Examples

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

# Use the iris data
data(iris)

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

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

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

# Make maps of each variable.

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

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


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

# Section 2: Create functions to predict species

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

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

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

# make lists of input and output map files.

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

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

# 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)


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

# map the predictions for the input maps
outfiles2 <- list(predict="spOutrf.txt")
```

```
AsciiGridPredict(rf,xfiles,outfiles2,xtypes=NULL,rows=NULL)

# read the asciigrids 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)))
sprf <- t(as.matrix(read.table("spOutrf.txt",skip=6)))
dist <- t(as.matrix(read.table("dist.txt",skip=6)))

par(mfcol=c(2,2))
image(spOrig,main="Original",axes=FALSE,useRaster=TRUE)
image(sprfNN,main="Using Predict",axes=FALSE,useRaster=TRUE)
image(sprf,main="Using Impute",axes=FALSE,useRaster=TRUE)
image(dist,main="Neighbor Distances",axes=FALSE,useRaster=TRUE)
```

`compare.yai` *Compares different k-NN solutions*

Description

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

Usage

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

Arguments

`...`	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.

Value

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

Author(s)

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

See Also

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

Examples

```
require(yaImpute)

data(iris)

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

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

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

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

---

cor.yai — *Correlation between observed and imputed*

Description

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

Usage

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

Arguments

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

Details

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

Value

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

Author(s)

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

References

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

See Also

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

`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

`object`	an object of class `yai` with `k > 1`.
`trgVal`	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.
`trgValCI`	The confidence interval that should contain the `mean(trgVal)`. If the mean falls within this interval, the problem is solved. If `NULL`, the interval is based on `nStdev`.
`nStdev`	the number of standard deviations in the vector of values used to compute the confidence interval when one is computed, ignored if `trgValCI` is not `NULL`.

`excludeRefIds`	identities of reference observations to exclude from the population, if coded as "all" then all references are excluded (see details).
`trace`	if `TRUE`, detailed output is produced.

Details

Imputation as it is defined in `yaImpute` can yield biased results. Lets 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. 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 the 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 nighbor 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 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 would 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 neighbors. The number of possible passes is equal to *k-1* where *k* is set in the original call to 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 excludeRefIds. When computing then mean of the attribute of interest (using the expression), 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 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 excludeRefIds argument as a vector of character strings identifing the rownames to leave out, or a vector of row numbers that identify the row numbers to leave out. If excludeRefIds="all", all reference observations are excluded.

Value

An object of class 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 yai and the call to this function. A new list, called biasParameters is added to the yai object with these tags:

trgValCI	the target CI.
curVal	the value of the bias that was achieved.
npasses	the number of passes through the data taken to achieve the result.
oldk	the old value of k.

Author(s)

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

See Also

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")

# find 5 refernece 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
```

`errorStats` *Compute error components of k-NN imputations*

Description

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

Usage

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

Arguments

`mahal`	An object of class `yai` computed with `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 `rmsdlg`).

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

Value

A list that contains several data frames. The column names of each are a combination of the name of the object used to compute the statistics and the name of the statistic. The rownames correspond 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: $sqrt(see^2 - (rmmsd0^2/2))$.

rmsd — root mean square error.

rmsdlg — root mean square error of the observations with larger distances.

sei — standard error of imputation $sqrt(rmsd^2 - (rmmsd0^2/2))$.

dstc — distance component: $sqrt(rmsd^2 - rmmsd0^2)$.

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

Author(s)

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

References

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

See Also

yai, TallyLake

Examples

```
require (yaImpute)

data(TallyLake)

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

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


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


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

errorStats(mal,msn)
```

---

foruse *Report a complete imputation*

Description

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

Usage

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

```
require(yaImpute)

data(iris)

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

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

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

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

---

impute.yai — *Impute variables from references to targets*

Description

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

Usage

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

Arguments

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`: an average over the *k* neighbors is taken;
`dstWeighted`: a weighted average is taken over the *k* neighbors where the weights are 1/(1+d).

`method.factor` the method used to compute the imputed values for factors, as follows:
`closest`: use the single neighbor that is closest (this is the default and is always used when *k*=1);
`mean`: actually is the *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 behaviour depends on the value of `ancillaryData`: when it is NULL, the Y-variables are imputed others all present in `ancillaryData` are imputed.

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

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

Scale factors for each variable are returned as an attribute (see `attributes`).

Author(s)

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

See Also

`yai`

Examples

```
require(yaImpute)

data(iris)
```

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

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

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

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

MoscowMtStJoe *Moscow Mountain and St. Joe Woodlands (Idaho, USA) Tree and 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:

Ground based measurements of trees:

- `ABGR_BA` Basal area (m^2/ha) of ABGR
- `ABLA_BA` Basal area (m^2/ha) of ABLA
- `ACGL_BA` Basal area (m^2/ha) of ACGL
- `BEOC_BA` Basal area (m^2/ha) of BEOC
- `LAOC_BA` Basal area (m^2/ha) of LAOC
- `PICO_BA` Basal area (m^2/ha) of PICO
- `PIEN_BA` Basal area (m^2/ha) of PIEN
- `PIMO_BA` Basal area (m^2/ha) of PIMO
- `PIPO_BA` Basal area (m^2/ha) of PIPO

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

Geographic Location, Slope and Aspect:

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

Advanced Land Imager (ALI):

- `B1MEAN` Mean of 30 m ALI band 1 pixels intersecting plot
- `B2MEAN` Mean of 30 m ALI band 2 pixels intersecting plot

- `B3MEAN` Mean of 30 m ALI band 3 pixels intersecting plot
- `B4MEAN` Mean of 30 m ALI band 4 pixels intersecting plot
- `B5MEAN` Mean of 30 m ALI band 5 pixels intersecting plot
- `B6MEAN` Mean of 30 m ALI band 6 pixels intersecting plot
- `B7MEAN` Mean of 30 m ALI band 7 pixels intersecting plot
- `B8MEAN` Mean of 30 m ALI band 8 pixels intersecting plot
- `B9MEAN` Mean of 30 m ALI band 9 pixels intersecting plot
- `PANMEAN`Mean of 10 m PAN band pixels intersecting plot
- `PANSTD` Standard deviation of 10 m PAN band pixels intersecting plot

LiDAR Intensity:

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

LiDAR Height:

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

LiDAR Canopy Cover:

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

Source

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

References

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

mostused *Tabulate references most often used in imputation*

Description

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

Usage

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

Arguments

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")

mostused(mal,kth=1)
```

newtargets *Finds K nearest neighbors for new target observations*

Description

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

Usage

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

Arguments

object	an object of class yai.
newdata	a data frame or matrix of new *targets* for which neighbors are are found.
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, which is a copy of the first argument with the following elements replaced:

call	the call.
obsDropped	a list of the row names for observations dropped for various reasons (missing data).
trgRows	a list of the row names for target observations as a subset of all observations.
xall	the *X*-variables for all observations.
neiDstTrgs	a data frame of distances between a target (identified by its row name) and the k references. There are k columns.
neiIdsTrgs	A data frame of reference identifications that correspond to 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.
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))

# 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 obervations 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

```
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`.

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 and 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")

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 thereshold 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>
Andrew O. Finley <finleya@msu.edu>

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

```
## 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*

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))
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)
```

`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)
`...`	passed to called methods, very useful for passing arugment `ancillaryData` to function `impute.yai`

Details

By default, RMSD is computed using standard formula for its related statistic, RMSE. When `scale=TRUE`, RMSD is divided by the standard deviation of the *reference* observations only 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`.

Author(s)

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

See Also

`yai`, `impute.yai` and `http://www.jstatsoft.org/v23/i10`.

TallyLake *Tally Lake, Flathead National Forest, Montana, USA*

Description

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

Usage

```
data(TallyLake)
```

Format

A data frame with 847 rows and 29 columns:

Ground based measurements of trees (Y-variables):

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

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

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

Additional X-variables:

- `ctim` Mean of slope curviture 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 curviture for pixels in stand
- `tancrvm` Mean of tangent curvature for pixels in stand
- `tancrvsd`Standard deviation of tangent curvature for pixels in stand

Source

USDA Forest Service

References

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

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

`unionDataJoin` *Combines data from several sources*

Description

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

Usage

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

Arguments

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

Value

A data frame.

Author(s)

Nicholas L. Crookston <ncrookston.fs@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
# 2     b  2
# 3     c  3
# 4     d  4
# 5     1  5
# 6     2  6
# 7     3  7
# 8     4  8
# 9  <NA>  9
# 10 <NA> 10

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

---

vars *List variables in a yai object*

Description

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

Usage

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

Arguments

`object`	an object created by `yai`.

Value

`yvars`	A charactor vector of *Y*-variables.
`xvars`	A charactor 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`

---

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 convereted to a factor.

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

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

Intended use is to transform community ecology data for use in `yai` where method is *randomForest*.

Usage

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

Arguments

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

Value

A data frame.

Author(s)

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

Examples

```
data(MoscowMtStJoe)

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

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

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

---

yai — *Find K nearest neighbors*

Description

Given a set of observations, yai

1. aeparates 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 referenece observations and between the reference and target observations.

An alternative method using randomForest classification and regression trees is provided for steps 2 and 3. *Target* observations are those with values for X-variables and not for Y-variables, while *reference* observations are those with no missing values for X-and Y-variables (see Details for the exception).

Usage

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

Arguments

- `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 `mahalanobis`. 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 finding neighbors; the options are the 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).
  - `gnn` distance is computed using a projected ordination of Xs found using canonical correspondence analysis (`cca` from package **vegan**).
  - `randomForest`distance is one minus the proportion of randomForest trees where a target observation is in the same terminal node as a reference observation (see `randomForest`).
  - `random`like raw except that the X space is a single vector of uniform random [0,1] numbers generated using `runif`, results in random assignement of neighbors, and forces `ann` to be FALSE.
- `ann` — TRUE if `ann` is used to find neighbors, FALSE if a slow search is used.
- `mtry` — the number of X-variables picked at random when method is `randomForest`, see `randomForest`, default is sqrt(number of X-variables).
- `ntree` — 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.

`rfMode` 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`.

Details

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

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

You need not have any Y-variables to run yai for the following methods: `euclidean`, `raw`, `mahalanobis`, `ica`, `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.

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.
`cancor`	returned from cancor function when method `msn` or `msn2` is used (NULL otherwise).
`ccaVegan`	an object of class cca (from package **vegan**) when method *gnn* 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.
`k`	the value of *k*.
`pVal`	as input; only used when method `msn` or `msn2` is used.
`projector`	NULL when not used. For methods *msn*, *msn2*, *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 data frame of distances between a target (identified by its row name) and the *k* references. There are *k* columns.
`neiIdsTrgs`	a data frame of reference identifications that correspond to neiDstTrgs.
`neiDstRefs, neiIdsRefs`	counterparts for references.

Author(s)

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

Examples

```
require (yaImpute)

data(iris)

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

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

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

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

data(MoscowMtStJoe)

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

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

```
x <- cbind(MoscowMtStJoe[,37:39],cartesian,MoscowMtStJoe[,42:64])
y <- MoscowMtStJoe[,1:35]

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

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

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

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

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

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

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

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

---

yaiRFsummary *Build Summary Data For Method RandomForest*

Description

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

Usage

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

Arguments

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`.

Author(s)

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

See Also

`yai`, `yaiVarImp`

`yaiVarImp` *Reprots or plots importance scores for method randomForest*

Description

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

Usage

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

Arguments

`object`	an object of class `yai`
`nTop`	the `nTop` most important variables are plotted (returned); if NA or zero, all are returned
`plot`	if FALSE, no plotting is done, but the scores are returned.
`...`	passed to boxplot function.

Value

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

Author(s)

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

See Also

yai, yaiRFsummary, compare.yai

Examples

```
data(MoscowMtStJoe)

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

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

yaiVarImp(rf)

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

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

yaiVarImp(rf2,col="gray")

compare.yai(rf,rf2)
```

Index