Package 'cabootcrs'

February 19, 2015

Type Package

Title Bootstrap Confidence Regions for Correspondence Analysis.

Version 1.0

Date 2013-04-23

Author T.J. Ringrose

Maintainer T.J. Ringrose <t.j.ringrose@cranfield.ac.uk>

Description Performs correspondence analysis on a two-way contingency table and produces bootstrap-based elliptical confidence regions around the projected coordinates for the category points. Includes routines to plot the results in a variety of styles. Also reports the standard numerical output for correspondence analysis.

License GPL-2

Imports methods, utils

LazyData TRUE

NeedsCompilation no

Repository CRAN

Date/Publication 2013-06-10 18:23:46

R topics documented:

`cabootcrs-package` *Bootstrap Confidence Regions for Correspondence Analysis*

Description

Performs simple (classical) correspondence analysis on a two-way contingency table and produces bootstrap-based confidence regions around the projected coordinates for the category points. Includes additional routines for summarising the output and for plotting the results in a variety of ways, including both french and biplot styles.

Details

Package:	cabootcrs
Type:	Package
Version:	1.0
Date:	2013-04-23
License:	GPL-2
Depends:	Base only

Correspondence Analysis plots usually only show the coordinates for each of the row and column category points projected onto the new axes, with no indication of the degree of sampling variation. This package produces bootstrap-based confidence ellipses for each of the row and column points with respect to the axes shown.

These confidence regions are based on the sampling variation of the difference between sample and population points when both are projected onto the sample axes, allowing for variation in both points and axes and the correlation between them. Hence the coverage percentage is the chance of drawing a sample such that the confidence ellipse contains the population point when it is projected onto the samples axes as a supplementary point. See the reference below for further details.

There are options for different ways of generating the bootstrap resamples, notably based on either the Poisson or the multinomial distribution, with the latter allowing the option of fixed row or column sums.

Correspondence analysis results can be plotted in two main ways. The default option here is to produce a biplot where the row category points are plotted in principal coordinates (i.e. coordinates which allow for the different inertias of the axes) and the confidence ellipses are shown for these row category points. The column category points are shown in standard coordinates on this plot and drawn as directions in the common biplot style. A second biplot is also produced where the roles of the rows and columns are reversed.

The other main plotting option is to produce a "french-style" plot where both row categories and column categories are plotted as points in principal coordinates. However, again two plots are produced, one with confidence ellipses for the row category points and one with confidence ellipses for the column category points. This is a deliberate restriction, partly to reduce plot clutter but mostly to emphasize that the row and column points are in different spaces and that their relative positions should not be over-interpreted.

There are several options for different ways of plotting the data, with simple options to vary the colour schemes, or to suppress point labels, or to show only a few of the ellipses, intended particularly to reduce the clutter in pictures with large data matrices. There is also an option for fuller control over the graphics, by supplying either files or data frames to define groups of points which can be plotted in common colours and symbols, or to suppress their point labels and ellipses.

The package can also be used just to perform Correspondence Analysis as usual, but with the above plotting options available.

The package does not use any routines from any of the many other Correspondence Analysis packages, only base R routines. This was deliberate, in order to maintain control over the precise details.

Author(s)

T.J. Ringrose <t.j.ringrose@cranfield.ac.uk>

References

Ringrose, T.J. (2012). Bootstrap confidence regions for correspondence analysis.
Journal of Statistical Computation and Simulation. Vol 83, No. 10, October 2012, 1397-1413.

See Also

cabootcrs , printca ,

plotca ,

summaryca , covmat , allvarscovs , cabootcrsresults

Examples

```
# Data frame, with row and column labels, from file

data(DreamData)

# Matrix with no labels

dreamdata <- t(matrix(c(7,4,3,7,10,15,11,13,23,9,11,7,28,9,12,10,32,5,4,3),4,5))

# Calculate variances and produce confidence ellipses for the Dream data set,
# with labels taken from those in the data file and default symbols and colours.
# Use all defaults: 1000 bootstraps, Poisson resampling, calculate variances
# only for first two axes, but give usual output for up to the first 4 axes.
# Show one biplot with confidence ellipses for row points in principal coordinates,
# another biplot with confidence ellipses for column points in principal coordinates.
# In each case the other set of points are in standard coordinates, but note that the
# lines are cropped to fit the plot by default, as it is the directions that matter most.

bd <- cabootcrs(DreamData)

# Same thing, but input data matrix rather than read from file,
# rows and columns by default just labelled by their number.

## Not run:
```

```
bd2 <- cabootcrs(dreamdata)

# Plot in "french" style where both rows and columns are in principal coordinates,
# not as a biplot, but still produce two plots, with row ellipses in one plot
# and column ellipses in the other.

plotca(bd, plottype="french")

# Calculate variances and covariances for axes 1-3, though only plots axis 1 versus 2.
# Then plot axis 1 against axis 3.

bd3 <- cabootcrs(DreamData, lastaxis=3)
plotca(bd3, firstaxis=1, lastaxis=3)

# See the stored results, an object of type cabootcrsresults.

bd

## End(Not run)

# Prettier printed output, no plots.

printca(bd)

# Brief summary output, similar style to ca package, no plots.

summaryca(bd,datasetname="Dreams")

# Extract the covariance matrix of:
# row 4 for axes 1 and 2;
# column 1 for axes 1 and 2.

vmr4 <- covmat(bd,4,"row",1,2)
vmc1 <- covmat(bd,1,"column",1,2)

# Display all variances and covariances for each row and column, axes 1-2.

allvarscovs(bd, "rows")
allvarscovs(bd, "columns")
```

`allvarscovs` *Extract variances and covariances*

Description

Extract a data frame containing all variances and covariances of either the row or the column points.

Usage

```
allvarscovs(x, thing = "rows")
```

Arguments

x	object of class cabootcrsresults.
thing	"rows" - all rows. "columns" - all columns.

Details

Shows all variances and covariances as a data frame for ease of viewing. To obtain a covariance matrix itself use covmat instead.

Value

Data frame.

Author(s)

T.J. Ringrose

See Also

covmat , cabootcrsresults

Examples

```
dreamdata <- t(matrix(c(7,4,3,7,10,15,11,13,23,9,11,7,28,9,12,10,32,5,4,3),4,5))
bd <- cabootcrs(dreamdata)
allvarscovs(bd)
allvarscovs(bd, thing="columns")


## The function is currently defined as
function (x, thing = "rows")
{
    getcovs <- function(allC, n, ncovs) {
        V <- matrix(0, n, ncovs)
        for (i in 1:n) {
            y <- allC[i, , ]
            V[i, ] <- y[upper.tri(y)]
        }
        invisible(V)
    }
    if (!(class(x) == "cabootcrsresults"))
        stop(paste("Must be of type cabootcrsresults\n\n"))
    if (!any(thing == c("rows", "columns")))
        stop(paste("Must be rows or columns\n\n"))
    ncovs <- x@axisvariances * (x@axisvariances - 1)/2
    vcnames <- character(length = x@axisvariances + ncovs)
    k <- 1
    for (i in 1:x@axisvariances) {
        vcnames[i] <- paste(" Var Axis", i)
        if (i < x@axisvariances) {
```

```
            for (j in (i + 1):x@axisvariances) {
                vcnames[x@axisvariances + k] <- paste(" Cov axes",
                  i, j)
                k <- k + 1
            }
        }
    }
    if (thing == "rows") {
        Covs <- getcovs(x@RowCov, x@rows, ncovs)
        allV <- data.frame(cbind(x@RowVar, Covs), row.names = x@rowlabels)
    }
    else {
        Covs <- getcovs(x@ColCov, x@columns, ncovs)
        allV <- data.frame(cbind(x@ColVar, Covs), row.names = x@collabels)
    }
    names(allV) <- vcnames
    allV
}
```

---

cabasicresults-class *Class* "cabasicresults"

Description

Contains all the fundamental output from the singular value decomposition at the heart of correspondence analysis, but nothing more.

Objects from the Class

Objects can be created by calls of the form `new("cabasicresults", ...)`.

Slots

Rprofile: Object of class "matrix" Row profile matrix

Cprofile: Object of class "matrix" Column profile matrix

Rweights: Object of class "matrix" Matrix of weights for row points: square roots of inverse column sums

Cweights: Object of class "matrix" Matrix of weights for column points: square roots of inverse row sums

Raxes: Object of class "matrix" Matrix of axes for row points: right singular vectors of weighted, centred data matrix

Caxes: Object of class "matrix" Matrix of axes for column points: left singular vectors of weighted, centred data matrix

r: Object of class "numeric" Rank of weighted, centred data matrix

mu: Object of class "numeric" Singular values of weighted, centred data matrix

Methods

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

Note

For internal use, not intended for direct user access.

Author(s)

T.J. Ringrose

See Also

cabootcrsresults

Examples

```
showClass("cabasicresults")
```

---

cabootcrs *Calculate category point variances using bootstrapping*

Description

Performs correspondence analysis and then uses bootstrap resampling to construct confidence ellipses for each row and column category point, printing and plotting the results.

For description of the package itself see cabootcrs-package.

Usage

```
cabootcrs(xobject = NULL, datafile = NULL,
     groupings = NULL, grouplabels = NULL,
     plotsymbolscolours=c(19,"alldifferent",18,"alldifferent"),
     othersmonochrome = "black", crpercent = 95, nboots = 1000,
     resampledistn = "Poisson", multinomialtype = "whole",
     poissonzeronewmean = 0, newzeroreset = 0,
     printdims = 4, lastaxis = 2,
     catype = "sca", bootstdcoords = FALSE)
```

Arguments

xobject	name of data object (data frame, matrix or similar class that can be coerced to data frame) containing the data in contingency table format. recommended that rows >= columns as matrix will be transposed if columns > rows.

`datafile` name of a text file (in " ") containing the data in contingency table format, with or without row and column labels.
ignored if xobject is non-null.
recommended that rows >= columns as matrix will be transposed if columns > rows.

`groupings` to be passed to plot, see `plotca` for details.

`grouplabels` to be passed to plot, see `plotca` for details.

`plotsymbolscolours`
to be passed to plot, see `plotca` for details.

`othersmonochrome`
to be passed to plot, see `plotca` for details.

`crpercent` to be passed to plot, see `plotca` for details.

`nboots` number of boostrap replicate matrices used, default and recommended minimum is 1000, but 10000 is recommended if machine and data set size allows; the calculated variances will sometimes differ around 3rd decimal place, but pictures should look the same.
if nboots=0 then correspondence analysis is performed as usual with no variances calculated.

`resampledistn` "Poisson" - resampled matrices constructed using Poisson resampling on each cell separately.
"multinomial" - resampled matrices constructed using multinomial resampling, treating the cells as defining one or more multinomial distributions.

`multinomialtype`
only relevant for multinomial sampling, otherwise ignored:
"whole" - all cells define a single multinomial distribution.
"rowsfixed" - each row defines a separate multinomial distribution.
"columnsfixed" - each column defines a separate multinomial distribution.

`poissonzeronewmean`
0 - no effect, method as described in paper.
> 0 - cells which are zero in the data are instead resampled from a Poisson distribution with this mean, which should be very small (say 0.1); this is for situations where rare cases did not occur but could have done, so that it might be appropriate for zero cells in the sample to be occasionally non-zero in resamples.

`newzeroreset` 0 - no effect, method as described in paper.
1 - if a cell value is non-zero in the sample but zero in the resample then it is reset to 1 in the resample, so that the sparsity structure of the sample is maintained in the resample. This can be useful with sparse data sets and, in effect, conditions on the sample sparsity structure.

`printdims` print full correspondence analysis coordinates, contributions, correlations etc for all output dimensions up to and including this one.

`lastaxis` calculate variances and covariances for all output axes (dimensions) up to this one (or the number of dimensions in the solution if smaller).
recommended maximum is 5 as variances for those above the fifth axis may be inaccurate.

`catype` "sca" - use simple (classical) correspondence analysis.
"mca", "omca" - not yet implemented

`bootstdcoords` FALSE - produce variances and confidence regions using principal coordinates, as in the paper.
TRUE - produce variances etc for standard coordinates.

Details

Performs all of the usual Correspondence Analysis calculations while also using bootstrapping to estimate, for each row and column category on each dimension of the solution, the variance of the difference between the sample and population point when both are projected onto the sample axes in principal coordinates, allowing for sampling variation in both the points and the axes.

Constructs confidence ellipses for each row and column category point and plots the results by a call to `plotca`. Also prints the usual Correspondence Analysis summary output and the calculated variances and covariances through calls to `summaryca` and `allvarscovs` respectively. Use `printca` for more detailed numerical results.

For further examples see `cabootcrs-package`.

Value

Object of class cabootcrsresults, plus output from calls to the summaryca, allvarscovs and plotca functions.

Note

In R itself both versions of the call to the routine, either with a data frame or matrix as input:

data(DreamData)
bd <- cabootcrs(DreamData)

or with a data file as input:

bd <- cabootcrs(datafile="DreamData.txt")

will work, assuming that setwd() has been used appropriately.

However, only the former worked in R CMD check when the package was being checked, so only that version appears in the examples.

Author(s)

T.J. Ringrose

References

Ringrose, T.J. (2012). Bootstrap confidence regions for correspondence analysis.
Journal of Statistical Computation and Simulation. Vol 83, No. 10, October 2012, 1397-1413.

See Also

`cabootcrs-package` , `printca` , `plotca` , `summaryca` , `covmat` , `allvarscovs` , `cabootcrsresults`

Examples

```
# Produce CRs for the Dream data set, supplied from a data frame.
# Use all defaults: 1000 bootstraps, Poisson resampling, calculate variances
# only for first two axes, but give usual output for up to the first 4 axes.
# Show a biplot with CRs for rows in principal coordinates and another with
# CRs for columns in principal coordinates.

data(DreamData)
bd <- cabootcrs(DreamData)

# Calculate variances and covariances for axes 1-3,
# though by default only plots axis 1 versus axis 2.

## Not run:
bd3 <- cabootcrs(DreamData, lastaxis=3)

# Use 10000 bootstrap replicates, which is recommended if the machine is
# fast enough but the difference is usually fairly small.

bd10k <- cabootcrs(DreamData, nboots=10000)

# Use multinomial resampling, with the matrix treated as a single
# multinomial distribution.

bdm <- cabootcrs(DreamData, resampledistn="multinomial")

# Fix the row sums (i.e. keep sum of age group constant).

bdmrf <- cabootcrs(DreamData, resampledistn="multinomial", multinomialtype="rowsfixed")

# Just perform correspondence analysis, without bootstrapping.

bd0 <- cabootcrs(DreamData, nboots=0)

## End(Not run)


## The function is currently defined as
function (xobject = NULL, datafile = NULL, groupings = NULL,
    grouplabels = NULL, plotsymbolscolours=c(19,"alldifferent",18,"alldifferent"),
    othersmonochrome = "black", crpercent = 95,
    nboots = 1000, resampledistn = "Poisson", multinomialtype = "whole",
    poissonzeronewmean = 0, newzeroreset = 0, printdims = 4,
    lastaxis = 2, catype = "sca", bootstdcoords = FALSE)
{
    if (nboots == 0) {
        cat(paste(
     "\n Standard Correspondence Analysis results only, no confidence regions calculated\n\n"))
    }
    else {
        if (nboots < 999)
            cat(paste("\n WARNING", nboots,
```

```
                    "is too few bootstrap replicates for reliable results\n\n"))
    if (!any(resampledistn == c("Poisson", "multinomial",
        "nonparametric", "balanced")))
      stop(paste("Resampling must be Poisson, multinomial, nonparametric or balanced\n\n"))
    if (!any(multinomialtype == c("whole", "rowsfixed", "columnsfixed")))
      stop(paste("Multinomial resampling is either whole, rowsfixed or columnsfixed\n\n"))
    if ((resampledistn == "Poisson") & any(multinomialtype ==
        c("rowsfixed", "columnsfixed")))
       cat(paste("\n WARNING can't fix rows or columns with Poisson resampling\n\n"))
    if (poissonzeronewmean < 0)
        stop(paste("Poisson mean for zero entries must be non-negative\n\n"))
    if (!any(newzeroreset == c(0, 1)))
       stop(paste("Zeros in sample can only be set to zero or one in bootstrap\n\n"))
}
if (printdims < 2)
    stop(paste("number of dims for output must be at least 2\n\n"))
if (lastaxis < 2)
    stop(paste("last axis must be at least 2\n\n"))
if (!any(catype == c("sca", "mca", "omca")))
    stop(paste("Must be sca, mca or omca"))
maxrearrange <- 6
if (is.null(xobject)) {
    Xtable <- read.table(file = datafile)
    if ((row.names(Xtable)[[1]] == "1") & (names(Xtable)[[1]] ==
        "V1")) {
        for (i in 1:dim(Xtable)[1]) rownames(Xtable)[i] <- paste("r",
            i, sep = "")
        for (i in 1:dim(Xtable)[2]) colnames(Xtable)[i] <- paste("c",
            i, sep = "")
    }
}
else {
    Xtable <- as.data.frame(xobject)
    if ((is.null(rownames(xobject))) | (row.names(Xtable)[[1]] ==
        "1")) {
        for (i in 1:dim(Xtable)[1]) rownames(Xtable)[i] <- paste("r",
            i, sep = "")
    }
    if ((is.null(colnames(xobject))) | (names(Xtable)[[1]] ==
        "V1")) {
        for (i in 1:dim(Xtable)[2]) colnames(Xtable)[i] <- paste("c",
            i, sep = "")
    }
}
X <- as.matrix(Xtable)
if ((dim(X)[2] > dim(X)[1]) & (catype == "sca")) {
    X <- t(X)
    rowlabels <- colnames(Xtable)
    collabels <- rownames(Xtable)
}
else {
    rowlabels <- rownames(Xtable)
    collabels <- colnames(Xtable)
```

```
}
rows <- dim(X)[1]
cols <- dim(X)[2]
n <- sum(X)
axisvariances <- min(lastaxis, rows - 1, cols - 1)
if ((lastaxis >= maxrearrange) & (cols >= maxrearrange +
    2))
    cat(paste("\n WARNING variances for the ", maxrearrange,
        "th axis and above may be unreliable\n\n"))
if (lastaxis>axisvariances)
    cat(paste("\n WARNING there are only", axisvariances,
        "axes in the solution\n\n"))
S <- switch(catype, sca = sca(X), mca = sca(X), omca = sca(X))
zerorowS <- rowSums(X > 0) == 0
zerocolS <- colSums(X > 0) == 0
onerowS <- rowSums(X > 0) == 1
onecolS <- colSums(X > 0) == 1
tworowS <- rowSums(X > 0) == 2
twocolS <- colSums(X > 0) == 2
RSBsum <- matrix(0, rows, axisvariances)
RSBsumsq <- matrix(0, rows, axisvariances)
RSBsumcp <- array(0, c(rows, axisvariances, axisvariances))
CSBsum <- matrix(0, cols, axisvariances)
CSBsumsq <- matrix(0, cols, axisvariances)
CSBsumcp <- array(0, c(cols, axisvariances, axisvariances))
rownB <- rep(nboots, rows)
colnB <- rep(nboots, cols)
RSBvar <- RSBsum
CSBvar <- CSBsum
RSBcov <- RSBsumcp
CSBcov <- CSBsumcp
sameaxisorder <- 0
if (resampledistn == "nonparametric") {
    bno <- matrix(sample(rep(1:rows, nboots), replace = TRUE),
        rows, nboots)
}
else {
    if (resampledistn == "balanced") {
        bno <- matrix(sample(rep(1:rows, nboots)), rows,
            nboots)
    }
}
for (b in 1:nboots) {
    if (resampledistn == "multinomial") {
        Xr <- vector("numeric", rows * cols)
        if (multinomialtype == "whole") {
            Xr <- rmultinom(1, n, X)
        }
        if (multinomialtype == "rowsfixed") {
            for (i in 1:rows) {
              Xr[seq(i, (cols - 1) * rows + i, by = rows)] <- rmultinom(1,
                sum(X[i, ]), X[i, ])
            }
```

```
    }
    if (multinomialtype == "columnsfixed") {
        for (i in 1:cols) {
          Xr[((i - 1) * rows + 1):(i * rows)] <- rmultinom(1,
            sum(X[, i]), X[, i])
        }
    }
}
else {
    if (resampledistn == "Poisson") {
        Xr <- rpois(rows * cols, X + poissonzeronewmean *
          (X == 0))
    }
    else {
        Xr <- X[bno[, b], ]
    }
}
XB <- matrix(data = Xr, nrow = rows, ncol = cols)
if (newzeroreset == 1) {
    XB <- (XB == 0 & X > 0) + XB
}
B <- switch(catype, sca = sca(XB), mca = sca(XB), omca = sca(XB))
zerorowB <- rowSums(XB > 0) == 0
zerocolB <- colSums(XB > 0) == 0
rownB <- rownB - zerorowB
colnB <- colnB - zerocolB
Re <- rearrange(S@Raxes, B@Raxes, S@Caxes, B@Caxes, B@r)
RaxesBRe <- B@Raxes
CaxesBRe <- B@Caxes
RaxesBRe[, 1:Re$numrearranged] <- RaxesBRe[, 1:Re$numrearranged] %*%
    Re$T
CaxesBRe[, 1:Re$numrearranged] <- CaxesBRe[, 1:Re$numrearranged] %*%
    Re$T
RSB <- (B@Rprofile - S@Rprofile) %*% B@Rweights %*% RaxesBRe[,
    1:axisvariances]
CSB <- (B@Cprofile - S@Cprofile) %*% B@Cweights %*% CaxesBRe[,
    1:axisvariances]
RSB <- RSB * (1 - zerorowS) * (1 - zerorowB)
CSB <- CSB * (1 - zerocolS) * (1 - zerocolB)
sameaxisorder <- sameaxisorder + Re$same
if (bootstdcoords == TRUE) {
    dmum1 <- diag(1/(B@mu + (B@mu == 0)) * (1 - (B@mu ==
        0)))
    dmum1[1:Re$numrearranged, 1:Re$numrearranged] <- dmum1[1:Re$numrearranged,
        1:Re$numrearranged] %*% Re$T
    dmum1 <- dmum1[1:axisvariances, 1:axisvariances]
    RSB <- RSB %*% dmum1
    CSB <- CSB %*% dmum1
}
RSBsum <- RSBsum + RSB
RSBsumsq <- RSBsumsq + RSB * RSB
CSBsum <- CSBsum + CSB
CSBsumsq <- CSBsumsq + CSB * CSB
```

```
    if (axisvariances > 1) {
        for (a1 in 1:(axisvariances - 1)) {
            for (a2 in (a1 + 1):axisvariances) {
              RSBsumcp[, a1, a2] <- RSBsumcp[, a1, a2] +
                RSB[, a1] * RSB[, a2]
              CSBsumcp[, a1, a2] <- CSBsumcp[, a1, a2] +
                CSB[, a1] * CSB[, a2]
            }
        }
    }
}
RSBmean <- diag((1/(rownB + (rownB == 0))) * (1 - (rownB ==
    0))) %*% RSBsum
CSBmean <- diag((1/(colnB + (colnB == 0))) * (1 - (colnB ==
    0))) %*% CSBsum
rbm1 <- diag((1/(rownB - 1 + 2 * (rownB <= 1))) * (1 - (rownB <=
    1)))
cbm1 <- diag((1/(colnB - 1 + 2 * (colnB <= 1))) * (1 - (colnB <=
    1)))
RSBvar <- rbm1 %*% (RSBsumsq - diag(rownB) %*% RSBmean *
    RSBmean)
CSBvar <- cbm1 %*% (CSBsumsq - diag(colnB) %*% CSBmean *
    CSBmean)
if (axisvariances > 1) {
    for (a1 in 1:(axisvariances - 1)) {
        for (a2 in (a1 + 1):axisvariances) {
            RSBcov[, a1, a2] <- rbm1 %*% (RSBsumcp[, a1,
              a2] - diag(rownB) %*% RSBmean[, a1] * RSBmean[,
              a2])
            CSBcov[, a1, a2] <- cbm1 %*% (CSBsumcp[, a1,
              a2] - diag(colnB) %*% CSBmean[, a1] * CSBmean[,
              a2])
        }
    }
}
Fmat <- S@Rprofile %*% S@Rweights %*% S@Raxes
Gmat <- S@Cprofile %*% S@Cweights %*% S@Caxes
dmum1 <- diag(1/(S@mu + (S@mu == 0)) * (1 - (S@mu == 0)))
Gbi <- Gmat %*% dmum1
Fbi <- Fmat %*% dmum1
inertia <- S@mu * S@mu
dmum2 <- diag(1/(inertia + (S@mu == 0)) * (1 - (S@mu == 0)))
inertiasum <- sum(inertia)
inertiapc <- 100 * inertia/inertiasum
cuminertiapc <- cumsum(inertiapc)
inertiapc <- round(100 * inertiapc)/100
cuminertiapc <- round(100 * cuminertiapc)/100
inertias <- cbind(inertia, inertiapc, cuminertiapc)
Xstd <- X/sum(X)
dr <- diag(as.vector(rowSums(Xstd)))
dc <- diag(as.vector(colSums(Xstd)))
Fsq <- Fmat * Fmat
RowCTR <- dr %*% Fsq %*% dmum2
```

```
  Frs <- diag(1/rowSums(Fsq))
  RowREP <- Frs %*% Fsq
  Gsq <- Gmat * Gmat
  ColCTR <- dc %*% Gsq %*% dmum2
  Grs <- diag(1/rowSums(Gsq))
  ColREP <- Grs %*% Gsq
  bootca <- new("cabootcrsresults", br = S, DataMatrix = X,
      rows = rows, columns = cols, rowlabels = rowlabels, collabels = collabels,
      Rowprincoord = Fmat, Colprincoord = Gmat, Rowstdcoord = Fbi,
      Colstdcoord = Gbi, RowCTR = RowCTR, RowREP = RowREP,
      ColCTR = ColCTR, ColREP = ColREP, RowVar = RSBvar, RowCov = RSBcov,
      ColVar = CSBvar, ColCov = CSBcov, inertiasum = inertiasum,
      inertias = inertias, nboots = nboots, resampledistn = resampledistn,
      multinomialtype = multinomialtype, sameaxisorder = sameaxisorder,
      poissonzeronewmean = poissonzeronewmean, newzeroreset = newzeroreset,
      printdims = printdims, axisvariances = axisvariances)
  summaryca(bootca, datasetname = as.character(datafile))
  plotca(bootca, datasetname = as.character(datafile), groupings = groupings,
      grouplabels = grouplabels, plotsymbolscolours = plotsymbolscolours,
      othersmonochrome = othersmonochrome, crpercent = crpercent)
  bootca
}
```

`cabootcrsresults-class` *Class* `"cabootcrsresults"`

Description

Contains an object of type cabasicresults, plus contributions and correlations etc, plus the variances and covariances calculated by bootstrapping, plus records the various settings used in doing this.

Objects from the Class

Objects can be created by calls of the form `new("cabootcrsresults", ...)`.

Slots

`br`: Object of class `"cabasicresults"` From the sample data matrix.

`DataMatrix`: Object of class `"matrix"` The sample data matrix.

`rows`: Object of class `"numeric"` Number of rows.

`columns`: Object of class `"numeric"` Number of columns.

`rowlabels`: Object of class `"character"` Row category labels.

`collabels`: Object of class `"character"` Column category labels.

`Rowprincoord`: Object of class `"matrix"` Principal coordinates for row points.

`Colprincoord`: Object of class `"matrix"` Principal coordinates for column points.

`Rowstdcoord`: Object of class `"matrix"` Standard coordinates for row points.

`Colstdcoord`: Object of class `"matrix"` Standard coordinates for column points.

`RowCTR`: Object of class `"matrix"` Contributions for row points.

`RowREP`: Object of class `"matrix"` Representations for row points.

`ColCTR`: Object of class `"matrix"` Contributions for column points.

`ColREP`: Object of class `"matrix"` Representations for column points.

`RowVar`: Object of class `"matrix"` Variances for row points.

`RowCov`: Object of class `"array"` Covariances for row points.

`ColVar`: Object of class `"matrix"` Variances for column points.

`ColCov`: Object of class `"array"` Covariances for column points.

`inertiasum`: Object of class `"numeric"` Total inertia.

`inertias`: Object of class `"matrix"` Axis inertias.

`nboots`: Object of class `"numeric"` Number of bootstrap replicates used to calculate the (co)variances.

`resampledistn`: Object of class `"character"` Distribution used for resampling.

`multinomialtype`: Object of class `"character"` Form of multinomial resampling used.

`sameaxisorder`: Object of class `"numeric"` Number of resamples with no reordering in first six bootstrap axes.

`poissonzeronewmean`: Object of class `"numeric"` Mean used for resampling zero cells.

`newzeroreset`: Object of class `"numeric"` Option to reset resample zero cells.

`printdims`: Object of class `"numeric"` Number of dimensions to print, though note that all are stored.

`axisvariances`: Object of class `"numeric"` Number of axes for which variances were calculated and are stored.

Methods

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

Note

This object class contains all the output from the cabootcrs routine. The generic routines print, plot and summary can be applied to produce different versions of the output without requiring further calculations.

The value in the axisvariances slot, giving the number of axes for which variances were calculated, is either the lastaxis parameter in the call to cabootcrs or the number of the dimensions in the solution, whichever is smaller.

Note that variances above the fifth axis may be inaccurate, see `rearrange` for further details.

Author(s)

T.J. Ringrose

See Also

cabasicresults , rearrange

Examples

```
showClass("cabootcrsresults")
```

---

covmat *Extract covariance matrix for a single category point*

Description

Extract the 2 by 2 covariance matrix, for one row or column point, for one pair of axes.

Usage

```
covmat(x, i, thing = "row", axis1 = 1, axis2 = 2)
```

Arguments

x	object of class cabootcrsresults.
i	row or column number.
thing	"row" - i-th row. "column" - i-th column.
axis1	first axis of pair.
axis2	second axis of pair, must be <= axisvariances value for x

Details

Extracts a covariance matrix for use in further calculations. To inspect variances and covariances use allvarscovs for more reader-friendly output.

Value

2 by 2 covariance matrix.

Author(s)

T.J. Ringrose

See Also

allvarscovs , cabootcrsresults

Examples

```
dreamdata <- t(matrix(c(7,4,3,7,10,15,11,13,23,9,11,7,28,9,12,10,32,5,4,3),4,5))
bd <- cabootcrs(dreamdata, lastaxis=3)
covrow3axes1and2 <- covmat(bd, i=3)
covcol2axes2and3 <- covmat(bd, i=2, thing="column", axis1=2, axis2=3)



## The function is currently defined as
function (x, i, thing = "row", axis1 = 1, axis2 = 2)
{
    printwithaxes <- function(res, thenames) {
        names(res) <- thenames
        print(res, digits = 4)
    }
    if (!(class(x) == "cabootcrsresults"))
        stop(paste("Must be of type cabootcrsresults\n\n"))
    if (!any(thing == c("row", "column")))
        stop(paste("Must be row or column\n\n"))
    if (axis1 == axis2)
        stop(paste("What are you playing at?\n\n"))
    if (!any(axis1 == seq(1, x@axisvariances)))
        stop(paste("Covariance not available for these axes\n\n"))
    if (!any(axis2 == seq(1, x@axisvariances)))
        stop(paste("Covariance not available for these axes\n\n"))
    if ((thing == "row") & !any(i == seq(1, x@rows)))
        stop(paste("Invalid row number\n\n"))
    if ((thing == "column") & !any(i == seq(1, x@columns)))
        stop(paste("Invalid column number\n\n"))
    a1 <- min(axis1, axis2)
    a2 <- max(axis1, axis2)
    tname <- ""
    if (thing == "row") {
        V <- matrix(c(x@RowVar[i, axis1], x@RowCov[i, a1, a2],
            x@RowCov[i, a1, a2], x@RowVar[i, axis2]), 2, 2)
        if (!is.null(x@rowlabels)) {
            tname <- paste("(", x@rowlabels[[i]], ")")
        }
    }
    else {
        V <- matrix(c(x@ColVar[i, axis1], x@ColCov[i, a1, a2],
            x@ColCov[i, a1, a2], x@ColVar[i, axis2]), 2, 2)
        if (!is.null(x@collabels)) {
            tname <- paste("(", x@collabels[[i]], ")")
        }
    }
    cat(paste("Covariance matrix of", switch(thing, row = "row",
        column = "column"), i, tname, "for axes", axis1, axis2,
        "\n\n"))
    rcnames <- c(paste("Axis", axis1), paste("Axis", axis2))
    printwithaxes(data.frame(V, row.names = rcnames), rcnames)
    invisible(V)
```

```
}
```

`DreamData` *Maxwell's Dream Data*

Description

A cross-classification of counts of boys by age-group (A=5-7, B=8-9, C=10-11, D=12-13, E=14-15) and severity of dream-disturbance (a=lowest, d=highest).

Usage

```
DreamData
```

Format

A contingency table.

Source

Iliopoulos et al.

References

G. Iliopoulos, M. Kateri and I. Ntzoufras (2007). *Bayesian estimation of unrestricted and order-restricted association models for a two-way contingency table*. Comput. Statist. Data Anal., 51, 4643-4655.

`plotca` *Produce correspondence analysis plots with confidence ellipses*

Description

Produces two plots of correspondence analysis results, one with confidence regions for row categories and one with confidence regions for column categories.

In the following, the categories for which confidence regions are being given are referred to as the primary points, the others as the secondary points. The primary points are always plotted in principal coordinates while the secondary points can be in standard (biplot style) or principal (french style) coordinates.

The default colour scheme is for the primary points and their confidence ellipses to be plotted each in a different colour, as this makes it easier to see which ellipse goes with which point, while the secondary points are all plotted in black to make it easier to distinguish between the two sets of points. This can all be changed.

Note that the plots will look better if saved as .eps or .pdf, rather than viewed in R or as .jpg or .png.

Usage

```
plotca(x, datasetname = "",
   showrowlabels = TRUE, showcolumnlabels = TRUE,
   groupings = NULL, grouplabels = NULL,
   plotsymbolscolours = c(19, "alldifferent", 18, "alldifferent"),
   othersmonochrome = "black", crpercent = 95, plottype = "biplot",
   showrowcrs = TRUE, showcolumncrs = TRUE,
   firstaxis = 1, lastaxis = 2, plotallpairs = FALSE,
   picsize = c(-1, 1))
```

Arguments

x	object of class cabootcrsresults.
datasetname	name of data set (in " "), to appear on plots.
showrowlabels	TRUE - label row points as usual. FALSE - suppress labels of row points.
showcolumnlabels	TRUE - label column points as usual. FALSE - suppress labels of column points.
groupings	name of file (in " ") or data frame containing group structure of row and column points; if the n rows are divided into m groups and the p columns divided into k groups, the file or data frame is n+p by 2, with the first column just 1..n 1..p (purely to make the file easier to read) and the second column containing the number of the group-of-rows (1..m) or group-of-columns (1..k) that the row or column belongs to: 1 < the number of the group-of-rows to which row 1 belongs > ... n < the number of the group-of-rows to which row n belongs > 1 < the number of the group-of-columns to which column 1 belongs > ... p < the number of the group-of-columns to which column p belongs >
grouplabels	name of file (in " ") or data frame containing the colours and labels to be used, in association with the groupings option above, in a m+k by 5 array: 1 < legend > < plot symbol > < plot colour > < draw ellipse? > ... m < legend > < plot symbol > < plot colour > < draw ellipse? > 1 < legend > < plot symbol > < plot colour > < draw ellipse? > ... k < legend > < plot symbol > < plot colour > < draw ellipse? > legend (in " " in data frame but not in file) - for this group of rows/columns, to be shown on plot plot symbol - for all rows/columns in this group, either an R number code for a symbol or a character (the latter in " " if in a data frame but not if in a file) colour (in " ") - for symbols and ellipses for this group

T - draw ellipses and label points for this group, F - suppress them (both in " " in data frame but not in file)
See notes section and examples below to make more sense of this.
These options are particularly intended for large data sets, to allow attention to be drawn to some points above others, to emphasize any group structure within the data, or to show only the most important ellipses in order to make the picture less cluttered.

`plotsymbolscolours`
: 4-vector in the form c(row symbol,"row colour", column symbol,"column colour") giving plot symbols and colours for row and column points and ellipses when they are the primary points.
First element:
any R number code - the plot symbol for row points.
character (in " ") - the plot symbol for row points.
Second element:
a valid R colour (in " ") - colour for row points, their ellipses and labels.
"alldifferent" - each row is plotted with a different colour, going through the R colour list with the first few rows as reds and the last few rows as blues.
"differentreds" - each row is plotted with a different shade of red to green.
"differentblues" - each row is plotted with a different shade of green to blue.
Third element:
as first element but for column points.
Fourth element:
as second element but for column points.

`othersmonochrome`
: a valid R colour (in " ") - all secondary points are plotted in this colour.
NULL - secondary points are plotted using the same colours as when they are the primary points.

`crpercent`
: notional coverage percentage of the confidence ellipses.

`plottype`
: "biplot" - one plot with confidence regions for rows in principal coordinates while columns are shown as directions in standard coordinates, another plot with confidence regions for columns in principal coordinates while rows are shown as directions in standard coordinates.
"french" - two plots each with both rows and columns in principal coordinates, one plot shows confidence regions for the rows and the other shows confidence regions for the columns.

`showrowcrs`
: TRUE - plot all confidence ellipses for row points as usual.
A row number or vector of row numbers - plot confidence ellipses only for these row points, which can be specified in any standard R format such as 7 or c(1,4,9) or 5:11.
FALSE - suppress plotting of all confidence ellipses for row points.

`showcolumncrs`
: TRUE - plot all confidence ellipses for column points as usual.
A column number or vector of column numbers - plot confidence ellipses only for these column points, which can be specified in any standard R format such as 7 or c(1,4,9) or 5:11.
FALSE - suppress plotting of all confidence ellipses for column points.

`firstaxis`	number of first (i.e. highest inertia) axis to be plotted.
`lastaxis`	number of last (i.e. lowest inertia) axis to be plotted, must be <= axisvariances value for x.
`plotallpairs`	FALSE - plot firstaxis v lastaxis only. TRUE - plot all pairs of axes between firstaxis and lastaxis.
`picsize`	if 2-vector, minimum and maximum of both x and y axes on each plot. if 4-vector, min and max of x axis followed by min and max of y axis. note that the same scales are used for both plots, so in biplot case it might occasionally be preferred to run plotca twice with different picsize values, one being better for rows in principal coordinates and the other better for columns in principal coordinates. note also that if picsize is used to focus in on a particular area of the plot then biplot labels might not appear properly.

Details

For large matrices the plots from exploratory multivariate methods are often so busy that the whole point of the method, to clarify the structure of the data, is nullified. This is even more of a problem when confidence regions are shown on the plots.

The showrowcrs, showcolumncrs, showrowlabels, showcolumnlabels and othersmonochrome options are available as ways of reducing plot clutter in large data sets, for example by showing the column points unlabelled and monochrome as a way of drawing the eye to the multicoloured row points and ellipses.

Note that french-style plots are often less cluttered because they omit the biplot lines, while they also show the two sets of points on similar scales so that it is easier to fit all the points on one picture without cropping or excessive empty space.

The groupings and grouplabels options are chosen via separate text files or data frames to define the groups of points, as it is usually easier to edit these than to edit lists of command options in R.

Value

Two plots with confidence regions shown.

Note

There are two ways of defining groupings and group labels. The first of these is by defining a pair of data frames within R and supplying them as parameters either to cabootcrs initially or to plotca. This method works in R CMD check and hence is the one used in the examples, but as you can see is rather hard to follow:

```
data(DreamData)
bd <- cabootcrs(DreamData)

groupingsframe <- cbind(c(1:5,1:4),c(1,1,2,2,3,1,1,2,2))
grouplabframe <- cbind( c(1,2,3,1,2), c("AB","CD","E","ab","cd"), c(19,20,21,"+","*"),
c("green","blue","yellow","red","orange"), "T" )
plotca(bd, groupings=groupingsframe, grouplabels=grouplabframe)
```

A version which produces identical results, but does not work in R CMD check, is usually much easier for the user. The groupings and group labels are defined in files, present in the directory specified in setwd(). To obtain identical results to the above, create two text files:

DreamGroupings.txt contains

1 1
2 1
3 2
4 2
5 3
1 1
2 1
3 2
4 2

e.g. the first two lines show that rows 1,2 belong to group-of-rows 1, while the last two lines show that columns 3,4 belong to group-of-columns 2.

DreamGroupLabels.txt contains

1 AB 19 "green" T
2 CD 20 "blue" T
3 E 21 "yellow" T
1 ab + "red" T
2 cd * "orange" T

e.g. group-of-rows 1 will be shown in green and plotted with symbol 19, with the legend AB.

These files can be edited outside R, which is usually much easier than doing things within R, and used in plotting with:

plotca(bd, groupings="DreamGroupings.txt", grouplabels="DreamGroupLabels.txt")

Note that plotca, summaryca and printca are all defined as new functions, rather than as overloaded versions of plot, summary and print, simply in order to avoid complication and unintended consequences within R.

Author(s)

T.J. Ringrose

See Also

printca , summaryca , cabootcrsresults

Examples

```
# the main function call plots with the default options

data(DreamData)
bd <- cabootcrs(DreamData)

## Not run:
# Plot with specified size to fit the whole of the arrows in without cropping
```

```
plotca(bd, picsize=c(-2.5,2.5))

# or smaller, note the warning

plotca(bd, picsize=c(-0.5,0.5))

# 90

plotca(bd, plotsymbolscolours=c(3,"differentreds","*","blue"), crpercent=90)

# suppress labels for column points, to de-clutter row points picture,
# this mostly useful for larger data sets than this one

plotca(bd, showcolumnlabels=FALSE, datasetname="Dream data")

# only show ellipses for rows 1, 1-2 and 1-3 respectively

plotca(bd, showrowcrs=1)
plotca(bd, showrowcrs=c(1,2))
plotca(bd, showrowcrs=1:3)


# both plots almost the same as the plot from the ca() package

plotca(bd,plottype="french",showrowcrs=FALSE,showcolumncrs=FALSE,othersmonochrome=NULL,
        plotsymbolscolours=c(19,"blue",17,"red"),picsize=c(-0.5,0.6))

# plot axes 1 v 2, 1 v 3 and 2 v 3

bd3 <- cabootcrs(DreamData, lastaxis=3)
plotca(bd3, firstaxis=1, lastaxis=3, plotallpairs=TRUE)

## End(Not run)

# more complicated plotting, define group structure in data frames

groupingsframe <- cbind(c(1:5,1:4),c(1,1,2,2,3,1,1,2,2))
grouplabframe <- cbind( c(1,2,3,1,2), c("AB","CD","E","ab","cd"), c(19,20,21,"+","*"),
                        c("green","blue","yellow","red","orange"), "T" )
plotca(bd, groupings=groupingsframe, grouplabels=grouplabframe)

## Not run:
plotca(bd, groupings=groupingsframe, grouplabels=grouplabframe, plottype="french")

## End(Not run)


## The function is currently defined as
function (x, datasetname = "", showrowlabels = TRUE, showcolumnlabels = TRUE,
    groupings = NULL, grouplabels = NULL, plotsymbolscolours = c(19,
        "alldifferent", 18, "alldifferent"), othersmonochrome = "black",
    crpercent = 95, plottype = "biplot", showrowcrs = TRUE, showcolumncrs = TRUE,
```

```
    firstaxis = 1, lastaxis = 2, plotallpairs = FALSE, picsize = c(-1,
        1))
{
    plotonepic <- function(a1, a2, plottype, things, nthings,
        nvars, Thingcoord, Varcoord, SBvar, SBcov, twoS, inertiapc,
        resampledistn, multinomialtype, thinggroup, thinggrlab,
        vargroup, vargrlab, thinglabels, varlabels, showcrs,
        picsizex, picsizey) {
        eps <- 1e-15
        critchisq2 <- qchisq(0.01 * crpercent, 2)
        critchisq1 <- qchisq(0.01 * crpercent, 1)
        theta <- seq(0, 2 * pi, 0.001)
        ellipsecoords <- rbind(sin(theta), cos(theta))
        thinggrlab3 <- as.list(thinggrlab[[3]])
        thinggrlab3int <- !is.na(as.integer(thinggrlab3))
        for (i in 1:max(thinggroup[, 2])) {
            if (thinggrlab3int[[i]]) {
                thinggrlab3[[i]] <- as.integer(thinggrlab3[[i]])
            }
        }
        if (plottype == "french") {
            vargrlab3 <- as.list(vargrlab[[3]])
            vargrlab3int <- !is.na(as.integer(vargrlab3))
            for (i in 1:max(vargroup[, 2])) {
                if (vargrlab3int[[i]]) {
                  vargrlab3[[i]] <- as.integer(vargrlab3[[i]])
                }
            }
        }
        dev.new()
        plot(Thingcoord[1, a1], Thingcoord[1, a2], xlim = picsizex,
            ylim = picsizey, xlab = paste("Axis ", a1, "    ",
                inertiapc[a1], "%", sep = ""), ylab = paste("Axis ",
                a2, "    ", inertiapc[a2], "%", sep = ""), asp = 1,
            pch = thinggrlab3[[thinggroup[1, 2]]], col = thinggrlab[[4]][thinggroup[1,
                2]])
        for (i in 2:nthings) {
            points(Thingcoord[i, a1], Thingcoord[i, a2], asp = 1,
              pch = thinggrlab3[[thinggroup[i, 2]]], col = thinggrlab[[4]][thinggroup[i,
                  2]])
        }
        abline(h = 0, v = 0)
        if (!all(thinggrlab[[2]] == "")) {
            labnum <- as.integer(thinggrlab3)
            labchar <- as.character(thinggrlab3)
            legend("topleft", thinggrlab[[2]], pch = labnum,
                col = thinggrlab[[4]], text.col = thinggrlab[[4]])
            for (i in 1:max(thinggroup[, 2])) {
                if (is.na(labnum[[i]])) {
                  labchar[[i]] <- thinggrlab3[[i]]
                }
                else {
                  labchar[[i]] <- NA
```

```
        }
    }
    legend("topleft", thinggrlab[[2]], pch = labchar,
        col = thinggrlab[[4]], text.col = thinggrlab[[4]])
}
if (plottype == "biplot") {
    if ((x@nboots > 0) & (any(showcrs == TRUE))) {
        title(paste("Confidence regions for biplot of",
          things, "\n \n", datasetname))
        title(paste("\n", resampledistn, "resampling,",
          switch(multinomialtype, whole = "", rowsfixed = "row sums fixed,",
            columnsfixed = "column sums fixed,"), x@nboots,
          "resamples \n"), font.main = 1)
    }
    else {
        title(paste("Biplot of", things, "\n", datasetname))
    }
    for (i in 1:nvars) {
        lines(c(0, Varcoord[i, a1]), c(0, Varcoord[i,
          a2]), col = vargrlab[[4]][vargroup[[2]][i]])
    }
    grat <- cbind(Varcoord[, a1]/picsizex[1], Varcoord[,
        a1]/picsizex[2], Varcoord[, a2]/picsizey[1],
        Varcoord[, a2]/picsizey[2], 0.95)/0.95
    cl <- 1.05/apply(grat, 1, max)
    text(cl * Varcoord[, a1], cl * Varcoord[, a2], labels = varlabels,
        col = vargrlab[[4]][vargroup[[2]]], pos = 4,
        cex = 0.75)
}
else {
    if ((x@nboots > 0) & (any(showcrs == TRUE))) {
        title(paste("Confidence regions for", things,
          "\n \n", datasetname))
        title(paste("\n", resampledistn, "resampling,",
          switch(multinomialtype, whole = "", rowsfixed = "row sums fixed,",
            columnsfixed = "column sums fixed,"), x@nboots,
          "resamples \n"), font.main = 1)
    }
    else {
        title(paste("Correspondence plot \n", datasetname))
    }
    for (i in 1:nvars) {
        points(Varcoord[i, a1], Varcoord[i, a2], asp = 1,
          pch = vargrlab3[[vargroup[i, 2]]], col = vargrlab[[4]][vargroup[i,
            2]])
    }
    text(Varcoord[, a1], Varcoord[, a2], labels = varlabels,
        col = vargrlab[[4]][vargroup[[2]]], pos = 4,
        cex = 0.75)
    if (!all(vargrlab[[2]] == "")) {
        labnum <- as.integer(vargrlab3)
        labchar <- as.character(vargrlab3)
        legend("topright", vargrlab[[2]], pch = labnum,
```

```
                col = vargrlab[[4]], text.col = vargrlab[[4]])
            for (i in 1:max(vargroup[, 2])) {
              if (is.na(labnum[[i]])) {
                labchar[[i]] <- vargrlab3[[i]]
              }
              else {
                labchar[[i]] <- NA
              }
            }
            legend("topright", vargrlab[[2]], pch = labchar,
              col = vargrlab[[4]], text.col = vargrlab[[4]])
        }
    }
    for (i in 1:nthings) {
        if (thinggrlab[[5]][thinggroup[i, 2]]) {
            text(Thingcoord[i, a1], Thingcoord[i, a2], labels = thinglabels[i],
              pos = 4, cex = 0.75, col = thinggrlab[[4]][thinggroup[i,
                2]])
            if (showcrs[i]) {
              xbar <- Thingcoord[i, cbind(a1, a2)]
              V <- matrix(c(SBvar[i, a1], SBcov[i, min(a1,
                a2), max(a1, a2)], SBcov[i, min(a1, a2),
                max(a1, a2)], SBvar[i, a2]), 2, 2)
              E <- eigen(V, symmetric = TRUE)
              usec2 <- (1 - twoS[i]) * (E$values[1] > eps)
              critchisq <- critchisq2 * usec2 + critchisq1 *
                (1 - usec2)
              coords <- E$vectors %*% (critchisq * diag(E$values))^(1/2) %*%
                ellipsecoords
              lines(xbar[1] + coords[1, ], xbar[2] + coords[2,
                ], pch = ".", col = thinggrlab[[4]][thinggroup[i,
                2]])
            }
        }
    }
    if (any(Thingcoord[, a1] < picsizex[1])) {
        cat(paste("Warning: point outside plot limits, lowest x-value is ",
            min(Thingcoord[, a1]), "\n"))
    }
    if (any(Thingcoord[, a1] > picsizex[2])) {
        cat(paste("Warning: point outside plot limits, largest x-value is ",
            max(Thingcoord[, a1]), "\n"))
    }
    if (any(Thingcoord[, a2] < picsizey[1])) {
        cat(paste("Warning: point outside plot limits, lowest y-value is ",
            min(Thingcoord[, a2]), "\n"))
    }
    if (any(Thingcoord[, a2] > picsizey[2])) {
        cat(paste("Warning: point outside plot limits, largest y-value is ",
            max(Thingcoord[, a2]), "\n"))
    }
}
if (!is.null(plotsymbolscolours)) {
```

```
    if (!dim(array(plotsymbolscolours)) == 4)
        stop(paste("plotsymbolscolours must contain row symbol and colour,
                    column symbol and colour\n\n"))
}
if (!any(plotsymbolscolours[c(2, 4)] == c(colours(), "alldifferent",
    "differentblues", "differentreds")))
    stop(paste("colours must be alldifferent, differentblues, differentreds
                or R colour (type colours() for full list) \n\n"))
if ((crpercent <= 0) | (crpercent >= 100))
    stop(paste("coverage percentage must be between 0 and 100 exclusive\n\n"))
if (!any(plottype == c("biplot", "french")))
    stop(paste("plotting must be biplot or french style\n\n"))
if (!any(class(showrowcrs) == c("integer", "numeric", "logical")))
    stop(paste("showrowcrs must be logical or a vector of row numbers\n\n"))
if (!any(class(showcolumncrs) == c("integer", "numeric",
    "logical")))
    stop(paste("showcolumncrs must be logical or a vector of row numbers\n\n"))
if ((firstaxis < 1) | (firstaxis > x@axisvariances - 1))
    stop(paste("incorrect first axis =", firstaxis, "\n\n"))
if (lastaxis > x@axisvariances)
    stop(paste("don't have variances for last axis =", lastaxis,
        "\n\n"))
if (firstaxis >= lastaxis)
    stop(paste("last axis must be greater than first axis\n\n"))
if (!any(dim(array(picsize)) == c(2, 4)))
  stop(paste("picsize bounds are  lower,upper OR x lower,x upper,y lower,y upper \n\n"))
if (picsize[1] >= picsize[2])
    stop(paste("incorrect axis scale picsize =", picsize[1],
        picsize[2], "\n\n"))
if (dim(array(picsize)) == 4) {
    if (picsize[3] >= picsize[4])
        stop(paste("incorrect y axis scale picsize =", picsize[3],
            picsize[4], "\n\n"))
    if (abs((picsize[4] - picsize[3]) - (picsize[2] - picsize[1])) >
        1e-10)
        stop(paste("x and y axes must be same length\n\n"))
}
options(warn = -1)
picsizey <- picsizex <- picsize[1:2]
if (dim(array(picsize)) == 4)
    picsizey <- picsize[3:4]
tworowS <- rowSums(x@DataMatrix > 0) == 2
twocolS <- colSums(x@DataMatrix > 0) == 2
if (is.null(groupings)) {
    if (any(plotsymbolscolours[2] == c("alldifferent", "differentreds",
        "differentblues"))) {
        rowgroup <- as.data.frame(cbind(1:x@rows, 1:x@rows))
        hv <- switch(plotsymbolscolours[2], alldifferent = c(0,
            0.85), differentreds = c(0, 0.45), differentblues = c(0.5,
            0.85))
        rowgrlab <- as.data.frame(cbind(1:x@rows, "", plotsymbolscolours[1],
            rainbow(n = x@rows, start = hv[1], end = hv[2]),
            "T"), stringsAsFactors = FALSE)
```

```
        }
        else {
            rowgroup <- as.data.frame(cbind(1:x@rows, rep(1,
                x@rows)))
            rowgrlab <- as.data.frame(cbind(1, "", plotsymbolscolours[1],
                plotsymbolscolours[2], "T"), stringsAsFactors = FALSE)
        }
        class(rowgrlab[, 1]) <- "integer"
        class(rowgrlab[, 5]) <- "logical"
        if (any(plotsymbolscolours[4] == c("alldifferent", "differentreds",
            "differentblues"))) {
            colgroup <- as.data.frame(cbind(1:x@columns, 1:x@columns))
            hv <- switch(plotsymbolscolours[4], alldifferent = c(0,
                0.85), differentreds = c(0, 0.45), differentblues = c(0.5,
                0.85))
            colgrlab <- as.data.frame(cbind(1:x@columns, "",
                plotsymbolscolours[3], rainbow(n = x@columns,
                  start = hv[1], end = hv[2]), "T"), stringsAsFactors = FALSE)
        }
        else {
            colgroup <- as.data.frame(cbind(1:x@columns, rep(1,
                x@columns)))
            colgrlab <- as.data.frame(cbind(1, "", plotsymbolscolours[3],
                plotsymbolscolours[4], "T"), stringsAsFactors = FALSE)
        }
        class(colgrlab[, 1]) <- "integer"
        class(colgrlab[, 5]) <- "logical"
    }
    else {
        if (class(groupings) == "character") {
            rcgroup <- read.table(file = groupings, colClasses = c("integer",
                "integer"))
        }
        else {
            rcgroup <- as.data.frame(groupings)
        }
        rowgroup <- rcgroup[1:x@rows, ]
        colgroup <- rcgroup[(x@rows + 1):(x@rows + x@columns),
            ]
        nrowgroups <- max(rowgroup[, 2])
        ncolgroups <- max(colgroup[, 2])
        if (class(grouplabels) == "character") {
            rcgrlab <- read.table(file = grouplabels, colClasses = c("integer",
                "character", "character", "character", "logical"))
        }
        else {
            rcgrlab <- as.data.frame(grouplabels, stringsAsFactors = FALSE)
            class(rcgrlab[, 1]) <- "integer"
            class(rcgrlab[, 5]) <- "logical"
        }
        rowgrlab <- rcgrlab[1:nrowgroups, ]
        colgrlab <- rcgrlab[(nrowgroups + 1):(nrowgroups + ncolgroups),
            ]
```

```
}
rowcrs <- logical(length = x@rows)
columncrs <- logical(length = x@columns)
if (any(class(showrowcrs) == c("numeric", "integer"))) {
    for (i in 1:length(showrowcrs)) {
        rowcrs[showrowcrs[i]] <- TRUE
    }
}
else {
    rowcrs <- rowcrs | showrowcrs
}
if (any(class(showcolumncrs) == c("numeric", "integer"))) {
    for (i in 1:length(showcolumncrs)) {
        columncrs[showcolumncrs[i]] <- TRUE
    }
}
else {
    columncrs <- columncrs | showcolumncrs
}
vrowgrlab <- rowgrlab
vcolgrlab <- colgrlab
if (any(othersmonochrome == colours())) {
    vrowgrlab[[4]] <- othersmonochrome
    vcolgrlab[[4]] <- othersmonochrome
}
if (showrowlabels == TRUE) {
    rowptlabels <- x@rowlabels
}
else {
    rowptlabels <- NULL
}
if (showcolumnlabels == TRUE) {
    colptlabels <- x@collabels
}
else {
    colptlabels <- NULL
}
for (a1 in firstaxis:(lastaxis - 1)) {
    for (a2 in (a1 + 1):lastaxis) {
        if ((plotallpairs == TRUE) | ((a1 == firstaxis) &
            (a2 == lastaxis))) {
            if (plottype == "biplot") {
              plotonepic(a1, a2, plottype, "rows", x@rows,
                x@columns, x@Rowprinccoord, x@Colstdcoord,
                x@RowVar, x@RowCov, tworowS, x@inertias[,
                  2], x@resampledistn, x@multinomialtype,
                rowgroup, rowgrlab, colgroup, vcolgrlab,
                rowptlabels, colptlabels, rowcrs, picsizex,
                picsizey)
              plotonepic(a1, a2, plottype, "columns", x@columns,
                x@rows, x@Colprinccoord, x@Rowstdcoord, x@ColVar,
                x@ColCov, twocolS, x@inertias[, 2], x@resampledistn,
                x@multinomialtype, colgroup, colgrlab, rowgroup,
```

```
                    vrowgrlab, colptlabels, rowptlabels, columncrs,
                    picsizex, picsizey)
                }
                else {
                  plotonepic(a1, a2, plottype, "rows", x@rows,
                    x@columns, x@Rowprinccoord, x@Colprinccoord,
                    x@RowVar, x@RowCov, tworowS, x@inertias[,
                      2], x@resampledistn, x@multinomialtype,
                    rowgroup, rowgrlab, colgroup, vcolgrlab,
                    rowptlabels, colptlabels, rowcrs, picsizex,
                    picsizey)
                  plotonepic(a1, a2, plottype, "columns", x@columns,
                    x@rows, x@Colprinccoord, x@Rowprinccoord,
                    x@ColVar, x@ColCov, twocolS, x@inertias[,
                      2], x@resampledistn, x@multinomialtype,
                    colgroup, colgrlab, rowgroup, vrowgrlab,
                    colptlabels, rowptlabels, columncrs, picsizex,
                    picsizey)
                }
            }
        }
    }
    options(warn = 0)
}
```

`printca` *Print full results*

Description

Prints full results from Correspondence Analysis, including variances, but no plots.

Usage

```
printca(x, datasetname = "")
```

Arguments

`x`	object of class cabootcrsresults.
`datasetname`	name of data set, to appear in output.

Details

Prints the usual Correspondence Analysis output plus the variances and covariances calculated by cabootcrs.

Firstly, the principal inertias for all dimensions.

Secondly, for the number of dimensions specified in the printdims slot, defined by the original call to cabootcrs:

Principal coordinates for row points
Contributions (per mil) for row points
Representations a.k.a. correlations (per mil) for row points
Principal coordinates for column points
Contributions (per mil) for column points
Representations a.k.a. correlations (per mil) for column points

Thirdly, for the number of dimensions defined by the axisvariances slot, which was defined by the lastaxis parameter in the original call to cabootcrs:

Estimated variances and covariances for row points.
Estimated variances and covariances for column points.

Value

Printed output.

Author(s)

T.J. Ringrose

See Also

plotca , summaryca , cabootcrsresults

Examples

```
dreamdata <- t(matrix(c(7,4,3,7,10,15,11,13,23,9,11,7,28,9,12,10,32,5,4,3),4,5))
bd <- cabootcrs(dreamdata)
printca(bd, datasetname="Dreams")


## The function is currently defined as
function (x, datasetname = "")
{
    printwithaxes <- function(res, thenames) {
        names(res) <- thenames
        print(res, digits = 4)
    }
    d <- min(x@printdims, x@br@r)
    axnames <- character(length = d)
    for (i in 1:d) {
        axnames[i] <- paste(" Axis", i)
    }
    cat("\n    RESULTS for Correspondence Analysis:", datasetname,
        "\n\n")
    cat("Total inertia ", x@inertiasum, "\n\n")
    cat("Inertias, percent inertias and cumulative percent inertias \n\n")
    ins <- data.frame(x@inertias)
    names(ins) <- c("Inertia", "%  ", "Cum. %")
    print(ins, digits = 6)
    cat("\nRows in principal coordinates\n\n")
    printwithaxes(data.frame(x@Rowprinccoord[, 1:d], row.names = x@rowlabels),
```

```
        axnames)
    cat("\nRow contributions (per mil)\n\n")
    printwithaxes(data.frame(round(x@RowCTR[, 1:d] * 1000), row.names = x@rowlabels),
        axnames)
    cat("\nRow representations (per mil)\n\n")
    printwithaxes(data.frame(round(x@RowREP[, 1:d] * 1000), row.names = x@rowlabels),
        axnames)
    cat("\nColumns in principal coordinates\n\n")
    printwithaxes(data.frame(x@Colprinccoord[, 1:d], row.names = x@collabels),
        axnames)
    cat("\nColumn contributions (per mil)\n\n")
    printwithaxes(data.frame(round(x@ColCTR[, 1:d] * 1000), row.names = x@collabels),
        axnames)
    cat("\nColumn representations (per mil)\n\n")
    printwithaxes(data.frame(round(x@ColREP[, 1:d] * 1000), row.names = x@collabels),
        axnames)
    if (x@nboots > 0) {
        cat("\n\n  Results for Bootstrapping\n\n")
        cat(x@nboots, "bootstrap replications with", x@resampledistn,
            "resampling\n")
        if (x@resampledistn == "multinomial" & x@multinomialtype !=
            "whole")
          cat(paste("  ", switch(x@multinomialtype, rowsfixed = "with row sums constant",
               columnsfixed = "with column sums constant"),
               "\n"))
        cat("\nEstimated variances and covariances\n\n")
        cat("Rows\n\n")
        print(allvarscovs(x, "rows"), digits = 4)
        cat("\nColumns\n\n")
        print(allvarscovs(x, "columns"), digits = 4)
        cat("\n\n")
    }
}
```

`rearrange` *Rearrange bootstrap axes by comparing to sample axes.*

Description

Compares one set of axes for row points and column points (from the bootstrap data matrix) to another (from the sample data matrix) by looking at all possible reorderings and reflections (only) of the bootstrap axes and picking the one which best matches the sample axes.

Usage

```
rearrange(RS, RB, CS, CB, r)
```

Arguments

RS	Sample axes for row points.
RB	Bootstrap axes for row points.
CS	Sample axes for column points.
CB	Bootstrap axes for column points.
r	Rank of bootstrap data matrix.

Details

Used to find the ordering of the bootstrap axes which best matches the sample axes under reordering and reflection, but not rotation.

Only the first six axes at most of the sample and bootstrap solutions are considered, for speed and simplicity. It is assumed that users are usually only interested in the first 2-4 axes of the sample solution and that hence the only reorderings of axes between sample and resample that are of interest are among the first six. Hence variances for the 6th axis may be inaccurate because reordering has not been fully allowed for, while those for the 7th axis and above will be very inaccurate.

Note that the routine is very literal and unsubtle and considers every possible ordering. The for loop calculating the match values is usually the main computational burden in the whole program, and a better algorithm for finding the best permutation for the bootstrap eigenvectors to match the sample eigenvectors would speed the program substantially.

Value

A list of items used in rearranging.

T	Matrix to postmultiply the bootstrap axes to match them to the sample axes.
numrearranged	Number of axes potentially rearranged = min(input rank,6).
match	Vector of values of the matching coefficient for each possible ordering.
same	TRUE if no reordering needed, FALSE otherwise.

Note

Internal routine, not intended for direct call by users.

Author(s)

T.J. Ringrose

See Also

sca , cabootcrs

Examples

```
## Not intended for direct call by users.

## The function is currently defined as
function (RS, RB, CS, CB, r)
{
    if (r >= 1) {
        maxrearrange <- 6
        numrearranged <- min(r, maxrearrange)
        switch(numrearranged, per <- matrix(1, 1, 1), per <- rbind(c(1,
            2), c(2, 1)), per <- cbind(rep(1:3, each = 2), c(2,
            3, 1, 3, 1, 2), c(3, 2, 3, 1, 2, 1)), {
            p <- cbind(rep(1:3, each = 2), c(2, 3, 1, 3, 1, 2),
                c(3, 2, 3, 1, 2, 1), 4)
            per <- rbind(p, p + 3 * (p == 1) - 3 * (p == 4),
                p + 2 * (p == 2) - 2 * (p == 4), p + (p == 3) -
                  (p == 4))
        }, {
            p <- cbind(rep(1:3, each = 2), c(2, 3, 1, 3, 1, 2),
                c(3, 2, 3, 1, 2, 1), 4)
            p <- rbind(p, p + 3 * (p == 1) - 3 * (p == 4), p +
                2 * (p == 2) - 2 * (p == 4), p + (p == 3) - (p ==
                4))
            p <- cbind(p, 5)
            per <- rbind(p, p + 4 * (p == 1) - 4 * (p == 5),
                p + 3 * (p == 2) - 3 * (p == 5), p + 2 * (p ==
                  3) - 2 * (p == 5), p + (p == 4) - (p == 5))
        }, {
            p <- cbind(rep(1:3, each = 2), c(2, 3, 1, 3, 1, 2),
                c(3, 2, 3, 1, 2, 1), 4)
            p <- rbind(p, p + 3 * (p == 1) - 3 * (p == 4), p +
                2 * (p == 2) - 2 * (p == 4), p + (p == 3) - (p ==
                4))
            p <- cbind(p, 5)
            p <- rbind(p, p + 4 * (p == 1) - 4 * (p == 5), p +
                3 * (p == 2) - 3 * (p == 5), p + 2 * (p == 3) -
                2 * (p == 5), p + (p == 4) - (p == 5))
            p <- cbind(p, 6)
            per <- rbind(p, p + 5 * (p == 1) - 5 * (p == 6),
                p + 4 * (p == 2) - 4 * (p == 6), p + 3 * (p ==
                  3) - 3 * (p == 6), p + 2 * (p == 4) - 2 * (p ==
                  6), p + (p == 5) - (p == 6))
        })
        nper <- dim(per)[1]
        match <- matrix(0, nper, 1)
        for (i in 1:nper) {
            match[i] = sum(diag(abs(t(RS[, 1:numrearranged]) %*%
                RB[, per[i, ]] + t(CS[, 1:numrearranged]) %*%
                CB[, per[i, ]])))
        }
        posn <- which.max(match)
        same <- posn == 1
```

```
        I <- diag(rep(1, numrearranged))
        T <- I[, per[posn, ]]
        t <- diag(t(RS[, 1:numrearranged]) %*% RB[, per[posn,
            ]] + t(CS[, 1:numrearranged]) %*% CB[, per[posn,
            ]])
        T <- T %*% diag((t >= 0) - (t < 0), nrow = numrearranged,
            ncol = numrearranged)
    }
    else {
        T <- matrix(1, 1, 1)
        numrearranged <- 1
        match <- 0
        same <- 0
    }
    list(T = T, numrearranged = numrearranged, match = match,
        same = same)
}
```

---

sca *Correspondence analysis on a contingency table or similar*

Description

Performs the usual simple (2-d) correspondence analysis calculations on an input data matrix, producing an object that contains the fundamental results but nothing more.

Usage

```
sca(X)
```

Arguments

X	A matrix of positive values, usually a contingency table.

Details

Assumes that input matrix has rows >= columns.

Value

An object of class cabasicresults.

Note

Internal routine, not intended for direct call by users, although it can be.

Author(s)

T.J. Ringrose

See Also

cabasicresults

Examples

```
dreamdata <- t(matrix(c(7,4,3,7,10,15,11,13,23,9,11,7,28,9,12,10,32,5,4,3),4,5))
cad <- sca(dreamdata)


## The function is currently defined as
function (X)
{
    X <- X/sum(X)
    rmax <- min(dim(X)) - 1
    rsums <- as.vector(rowSums(X))
    csums <- as.vector(colSums(X))
    drm1 <- diag(1/(rsums + (rsums == 0)) * (1 - (rsums == 0)))
    dcm1 <- diag(1/(csums + (csums == 0)) * (1 - (csums == 0)))
    drmh <- sqrt(drm1)
    dcmh <- sqrt(dcm1)
    Z <- drmh %*% (X - rsums %*% t(csums)) %*% dcmh
    Y <- svd(Z)
    mu <- Y$d[1:rmax]
    r <- sum(mu > 1e-15)
    R <- drm1 %*% X
    Rweights <- dcmh
    Raxes = Y$v[, 1:rmax]
    C <- dcm1 %*% t(X)
    Cweights <- drmh
    Caxes = Y$u[, 1:rmax]
    if (r < rmax) {
        mu[(r + 1):rmax] <- 0
        Raxes[, (r + 1):rmax] <- 0
        Caxes[, (r + 1):rmax] <- 0
    }
    sca <- new("cabasicresults", Rprofile = R, Cprofile = C,
        Rweights = Rweights, Cweights = Cweights, Raxes = Raxes,
        Caxes = Caxes, r = r, mu = mu)
  }
```

---

summaryca *Summarise results*

Description

Produces brief printed output of the usual correspondence analysis results for the first two dimensions of the solution, plus the standard deviations.

Usage

```
summaryca(x, datasetname = "")
```

Arguments

x	object of class cabootcrsresults
datasetname	name of data set, to appear in output

Details

Gives the principal inertias for all dimensions, followed by, for rows and then columns:

Principal coordinate, first axis
Standard deviation, first axis
Representation, a.k.a. correlation (per mil), first axis
Contribution (per mil), first axis

Principal coordinate, second axis
Standard deviation, second axis
Representation (per mil), second axis
Contribution (per mil), second axis

Representation, a.k.a. correlation (per mil), first two axes

Value

Printed summary output.

Author(s)

T.J. Ringrose

See Also

printca , plotca , cabootcrsresults

Examples

```
dreamdata <- t(matrix(c(7,4,3,7,10,15,11,13,23,9,11,7,28,9,12,10,32,5,4,3),4,5))
bd <- cabootcrs(dreamdata)
summaryca(bd, datasetname="Dreams")


## The function is currently defined as
function (x, datasetname = "")
{
    colnames <- character(length = 9)
    colnames <- c("  Axis 1", "StDev", "Rep", "Ctr", "  Axis 2",
        "StDev", "Rep", "Ctr", "Quality")
    colnamesnosd <- character(length = 7)
    colnamesnosd <- c("  Axis 1", "Rep", "Ctr", "  Axis 2", "Rep",
        "Ctr", "Quality")
```

```
  cat("\n    SUMMARY RESULTS for Correspondence Analysis:",
      datasetname, "\n\n")
  cat("Total inertia ", x@inertiasum, "\n\n")
  cat("Inertias, percent inertias and cumulative percent inertias \n\n")
  ins <- data.frame(x@inertias)
  names(ins) <- c("Inertia", "%  ", "Cum. %")
  print(ins, digits = 4)
  cat("\n")
  if (x@nboots > 0) {
      cat("Princ coords, std devs; rep and ctr (per mil); 2-d rep (per mil)\n\n")
  }
  else {
      cat("Princ coords; rep and ctr (per mil); 2-d rep (per mil)\n\n")
  }
  cat("Rows: \n")
  rop <- data.frame(round(x@Rowprinccoord[, 1] * 1000)/1000,
      round(sqrt(x@RowVar[, 1]) * 1000)/1000, round(x@RowREP[,
          1] * 1000), round(x@RowCTR[, 1] * 1000), round(x@Rowprinccoord[,
          2] * 1000)/1000, round(sqrt(x@RowVar[, 2]) * 1000)/1000,
      round(x@RowREP[, 2] * 1000), round(x@RowCTR[, 2] * 1000),
      round(rowSums(x@RowREP[, 1:2] * 1000)), row.names = x@rowlabels)
  if (x@nboots == 0) {
      rop <- rop[, c(1, 3, 4, 5, 7, 8, 9)]
      names(rop) <- colnamesnosd
  }
  else {
      names(rop) <- colnames
  }
  print(rop, digits = 3)
  cat("\n")
  cat("Columns: \n")
  cop <- data.frame(round(x@Colprinccoord[, 1] * 1000)/1000,
      round(sqrt(x@ColVar[, 1]) * 1000)/1000, round(x@ColREP[,
          1] * 1000), round(x@ColCTR[, 1] * 1000), round(x@Colprinccoord[,
          2] * 1000)/1000, round(sqrt(x@ColVar[, 2]) * 1000)/1000,
      round(x@ColREP[, 2] * 1000), round(x@ColCTR[, 2] * 1000),
      round(rowSums(x@ColREP[, 1:2] * 1000)), row.names = x@collabels)
  if (x@nboots == 0) {
      cop <- cop[, c(1, 3, 4, 5, 7, 8, 9)]
      names(cop) <- colnamesnosd
  }
  else {
      names(cop) <- colnames
  }
  print(cop, digits = 3)
  cat("\n")
}
```

Index