Package ‘Anthropometry’

October 11, 2021

Type Package

Title Statistical Methods for Anthropometric Data

Version 1.16

Date 2021-10-11

Author Guillermo Vinue, Irene Epifanio, Amelia Simo, M. Victoria Ibanez, Juan Domingo, Guillermo Ayala

Maintainer Guillermo Vinue <Guillermo.Vinue@uv.es>

Description Statistical methodologies especially developed to analyze anthropometric data. These methods are aimed at providing effective solutions to some commons problems related to Ergonomics and Anthropometry. They are based on clustering, the statistical concept of data depth, statistical shape analysis and archetypal analysis. Please see Vinue (2017) <doi:10.18637/jss.v077.i06>.

License GPL (>= 2)


Depends R (>= 3.3.0)

Imports shapes, rgl, archetypes, nnls, depth, FNN, ICGE, cluster, biclust

Suggests knitr, calibrate, mvtnorm, SportsAnalytics, RColorBrewer, plotrix, abind

VignetteBuilder knitr

LazyData yes

NeedsCompilation yes

Repository CRAN

Date/Publication 2021-10-11 08:50:05 UTC

R topics documented:

Anthropometry-package .................................................. 3
anthrCases .............................................................. 5
archetypesBoundary ..................................................... 6
<table>
<thead>
<tr>
<th>R topics documented:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>archetypoids</td>
<td>9</td>
</tr>
<tr>
<td>array3Dlandm</td>
<td>12</td>
</tr>
<tr>
<td>bustSizesStandard</td>
<td>13</td>
</tr>
<tr>
<td>CCbiclustAnthropo</td>
<td>14</td>
</tr>
<tr>
<td>cdfDissWomenPrototypes</td>
<td>17</td>
</tr>
<tr>
<td>checkBranchLocalIMO</td>
<td>19</td>
</tr>
<tr>
<td>checkBranchLocalMO</td>
<td>21</td>
</tr>
<tr>
<td>computSizesHipamAnthropom</td>
<td>23</td>
</tr>
<tr>
<td>computSizesTrimowa</td>
<td>24</td>
</tr>
<tr>
<td>cube34landm</td>
<td>25</td>
</tr>
<tr>
<td>cube8landm</td>
<td>26</td>
</tr>
<tr>
<td>descrDissTrunks</td>
<td>27</td>
</tr>
<tr>
<td>figures8landm</td>
<td>28</td>
</tr>
<tr>
<td>getBestPamsamIMO</td>
<td>29</td>
</tr>
<tr>
<td>getBestPamsamMO</td>
<td>30</td>
</tr>
<tr>
<td>getDistMatrix</td>
<td>32</td>
</tr>
<tr>
<td>HartiganShapes</td>
<td>34</td>
</tr>
<tr>
<td>hipamAnthropom</td>
<td>37</td>
</tr>
<tr>
<td>landmarksSampleSpaSurv</td>
<td>40</td>
</tr>
<tr>
<td>LloydShapes</td>
<td>41</td>
</tr>
<tr>
<td>nearestToArchetypes</td>
<td>43</td>
</tr>
<tr>
<td>optraShapes</td>
<td>44</td>
</tr>
<tr>
<td>overlapBiclustersByRows</td>
<td>46</td>
</tr>
<tr>
<td>parallelep34landm</td>
<td>48</td>
</tr>
<tr>
<td>parallelep8landm</td>
<td>49</td>
</tr>
<tr>
<td>percentilsArchetypoid</td>
<td>49</td>
</tr>
<tr>
<td>plotPrototypes</td>
<td>51</td>
</tr>
<tr>
<td>plotTreeHipamAnthropom</td>
<td>53</td>
</tr>
<tr>
<td>plotTrimmOutl</td>
<td>55</td>
</tr>
<tr>
<td>preprocessing</td>
<td>57</td>
</tr>
<tr>
<td>projShapes</td>
<td>58</td>
</tr>
<tr>
<td>qtranShapes</td>
<td>59</td>
</tr>
<tr>
<td>sampleSpanishSurvey</td>
<td>61</td>
</tr>
<tr>
<td>screeArchetypal</td>
<td>62</td>
</tr>
<tr>
<td>shapes3dShapes</td>
<td>66</td>
</tr>
<tr>
<td>skeletonsArchetypal</td>
<td>67</td>
</tr>
<tr>
<td>stepArchetypesRawData</td>
<td>68</td>
</tr>
<tr>
<td>stepArchetypoids</td>
<td>70</td>
</tr>
<tr>
<td>TDDclust</td>
<td>72</td>
</tr>
<tr>
<td>trimmedLloydShapes</td>
<td>74</td>
</tr>
<tr>
<td>trimmedoid</td>
<td>76</td>
</tr>
<tr>
<td>trimmOutl</td>
<td>78</td>
</tr>
<tr>
<td>trimowa</td>
<td>80</td>
</tr>
<tr>
<td>USAFSurvey</td>
<td>82</td>
</tr>
<tr>
<td>weightsMixtureUB</td>
<td>83</td>
</tr>
<tr>
<td>xyplotPCArchetypes</td>
<td>84</td>
</tr>
</tbody>
</table>

| Index   | 86 |
Description

Statistical methodologies especially developed to analyze anthropometric data. These methods are aimed at providing effective solutions to some commons problems related to Ergonomics and Anthropometry. They are based on clustering, the statistical concept of data depth, statistical shape analysis and archetypal analysis. Please see Vinue (2017) <doi:10.18637/jss.v077.i06>.

Details

Package: Anthropometry
Type: Package
Version: 1.15
Date: 2021-04-18
License: GPL (>=2)
LazyLoad: yes
LazyData: yes

anthrCases: Helper generic function for obtaining the anthropometric cases.
Anthropometry-internalArchetypoids: Several internal functions to compute and represent archetypes and archetypoids.
Anthropometry-internalHipamAnthropom: Several internal functions used by both $HIPAM_MO$ and $HIPAM_IMO$ algorithms.
Anthropometry-internalPlotTree: Several internal functions used to build the HIPAM plot tree.
Anthropometry-internalTDDclust: Several internal functions to clustering based on the L1 data depth.
archetypesBoundary: Archetypal analysis in multivariate accommodation problem.
archetypoids: Finding archetypoids.
array3Dlandm: Helper function for the 3D landmarks.
bustSizesStandard: Helper function for defining the bust sizes.
CCbiclustAnthropo: Cheng and Church biclustering algorithm applied to anthropometric data.
cdfDissWomenPrototypes: CDF for the dissimilarities between women and computed medoids and standard prototypes.
checkBranchLocalIMO: Evaluation of the candidate clustering partition in $HIPAM_IMO$.
checkBranchLocalMO: Evaluation of the candidate clustering partition in $HIPAM_MO$.
computSizesTrimowa: Computation of the trimowa elements for a given number of sizes defined by the EN.
computSizesHipamAnthropom: Computation of the hipamAnthropom elements for a given number of sizes defined by the EN.
cube8landm: Cube of 8 landmarks.
cube34landm: Cube of 34 landmarks.
descrDissTrunks: Description of the dissimilarities between women’s trunks.
figures8landm: Figures of 8 landmarks with labelled landmarks.
Anthropometry-package

getBestPamsamIMO: Generation of the candidate clustering partition in HIPAM.IMO.
getBestPamsamMO: Generation of the candidate clustering partition in HIPAM.MO.
getDistMatrix: Dissimilarity matrix between individuals and prototypes.
HartiganShapes: Hartigan-Wong k-means for 3D shapes.
hipamAnthropom: HIPAM algorithm for anthropometric data.
landmarksSampleSpaSurv: Landmarks of the sampled women of the Spanish Survey.
LloydShapes: Lloyd k-means for 3D shapes.
nearestToArchetypes: Nearest individuals to archetypes.
overlapBiclustersByRows: Overlapped biclusters by rows.
parallelep8landm: Parallelepiped of 8 landmarks.
parallelep34landm: Parallelepiped of 34 landmarks.
percentilsArchetypoid: Helper function for computing percentiles of a certain archetypoid.
plotPrototypes: Prototypes representation.
plotTreeHipamAnthropom: HIPAM dendogram.
plotTrimOutl: Trimmed or outlier observations representation.
preprocessing: Data preprocessing before computing archetypal observations.
projShapes: Helper function for plotting the shapes.
qtranShapes: Auxiliary qtran subroutine of the Hartigan-Wong k-means for 3D shapes.
sampleSpanishSurvey: Sample database of the Spanish anthropometric survey.
screeArchetypal: Screeplot of archetypal individuals.
shapes3dShapes: 3D shapes plot.
skeletonsArchetypal: Skeleton plot of archetypal individuals.
stepArchetypesRawData: Archetype algorithm to raw data.
stepArchetypoids: Run the archetypoid algorithm several times.
TDDclust: Trimmed clustering based on L1 data depth.
trimmedLloydShapes: Trimmed Lloyd k-means for 3D shapes.
trimmedmedoid: Trimmed k-medoids algorithm.
trimmOutl: Helper generic function for obtaining the trimmed and outlier observations.
trimowa: Trimmed PAM with OWA operators.
weightsMixtureUB: Calculation of the weights for the OWA operators.
xyplotPCArchetypes: PC scores for archetypes.

Author(s)

Guillermo Vinue <Guillermo.Vinue@uv.es>, Irene Epifanio, Amelia Simo, M. Victoria Ibanez, Juan Domingo, Guillermo Ayala

References

The `anthrCases` function is a helper generic function for obtaining the anthropometric cases. It is used to identify the cases computed by each method in an easy way. The usage is as follows:

```r
anthrCases(resMethod, nsizes)
## S3 method for class 'trimowa'
anthrCases(resMethod, nsizes)
## S3 method for class 'hipamAnthropom'
anthrCases(resMethod, nsizes)
```

The arguments are:

- `resMethod`: This is the object which saves the results obtained by the methodologies and which contains the anthropometric cases to return.
- `nsizes`: Number of bust sizes. This argument is needed for the "trimowa" and "hipamAnthropom" methodologies because they can compute the prototypes for any given number of bust sizes.

The value returned is a vector of class `anthrCases` with the anthropometric cases.

**Author(s)**

Guillermo Vinue

**References**


archetypesBoundary

See Also
trimowa, TDDclust, hipamAnthropom, LloydShapes, HartiganShapes, trimmedLloydShapes, archetypoids, stepArchetypoids

Examples

```r
# kmeans Procrustes:
landmarksNoNa <- na.exclude(landmarksSampleSpaSurv)
dim(landmarksNoNa)
# [1] 574 198
numLandmarks <- (dim(landmarksNoNa)[2]) / 3
# [1] 66
# As a toy example, only the first 10 individuals are used.
landmarksNoNa_First10 <- landmarksNoNa[1:10, ]
(numIndiv <- dim(landmarksNoNa_First10)[1])
# [1] 10
array3D <- array3Dlandm(numLandmarks, numIndiv, landmarksNoNa_First10)
# shapes::plotshapes(array3D[, , 1])
# calibrate::textxy(array3D[, 1, 1], array3D[, 2, 1], labs = 1:numLandmarks, cex = 0.7)
numClust <- 2; algSteps <- 1; niter <- 1; stopCr <- 0.0001
resLL <- LloydShapes(array3D, numClust, algSteps, niter, stopCr, FALSE, FALSE)
prototypes <- anthrCases(resLL)
```

archetypesBoundary  Archetypal analysis in multivariate accommodation problem

Description

This function allows us to reproduce the results shown in section 2.2.2 and section 3.1 of Epifanio et al. (2013). In addition, from the results provided by this function, the other results shown in section 3.2 and section 3.3 of the same paper can be also reproduced (see section examples below).

Usage

```r
archetypesBoundary(data, numArch, verbose, numRep)
```

Arguments

data  USAF 1967 database (see USAFSurvey). Each row corresponds to an observation, and each column corresponds to a variable. All variables are numeric.
numArch  Number of archetypes (archetypal observations).
verbose  Logical value. If TRUE, some details of the execution progress are shown (this is the same argument as that of the stepArchetypes function of the archetypes R package (Eugster (2009))).
numRep  For each archetype run archetypes numRep times (this is the same argument as the nrep argument of the stepArchetypes function of archetypes).
Details

Before using this function, the more extreme (100 - percAcomm*100)% observations must be removed by means of the preprocessing function. To that end, it is recommended that you use the Mahalanobis distance. In this case, the depth procedure has the disadvantage that the desired percentage of accommodation is not under control of the analyst and it may not exactly coincide with that one indicated.

Value

A list with numArch elements. Each element is a list of class attribute stepArchetypes with numRep elements.

Note

We would like to note that, some time after publishing the paper Epifanio et al. (2013), we found out that the stepArchetypes function standardizes the data by default (even when the data are already standardized) and this option is not always desired. In order to avoid this way of proceeding, we have created the stepArchetypesRawData function, which is used within archetypesBoundary instead of using stepArchetypes. Therefore, the results provided by archetypesBoundary allows us to reproduce the results of Epifanio et al. (2013) but they are now slightly different.

Author(s)

Irene Epifanio and Guillermo Vinue

References


See Also

archetypes, stepArchetypes, stepArchetypesRawData, USAFSurvey, nearestToArchetypes, preprocessing

Examples

#The following R code allows us to reproduce the results of the paper Epifanio et al. (2013).
#As a toy example, only the first 25 individuals are used.
#First, the USAF 1967 database is read and preprocessed (Zehner et al. (1993)).
#Variable selection:
#variable_sel <- c(48, 40, 39, 33, 34, 36)
#Changing to inches:
USAFSurvey_inch <- USAFSurvey[1:25, variabl_sel] / (10 * 2.54)

# Data preprocessing:
USAFSurvey_preproc <- preprocessing(USAFSurvey_inch, TRUE, 0.95, TRUE)

# Procedure and results shown in section 2.2.2 and section 3.1:
# For reproducing results, seed for randomness:
# suppressWarnings(RNGversion("3.5.0"))
# set.seed(2010)
res <- archetypesBoundary(USAFSurvey_preproc$data, 15, FALSE, 3)
# To understand the warning messages, see the vignette of the
# archetypes package.

# Results shown in section 3.2 (figure 3):
screeplot(res)

# 3 archetypes:
a3 <- archetypes::bestModel(res[3])
archetypes::parameters(a3)
# 7 archetypes:
a7 <- archetypes::bestModel(res[7])
archetypes::parameters(a7)
# Plotting the percentiles of each archetype:
# Figure 2 (b):
barplot(a3,USAFSurvey_preproc$data, percentiles = TRUE, which = "beside")
# Figure 2 (f):
barplot(a7,USAFSurvey_preproc$data, percentiles = TRUE, which = "beside")

# Results shown in section 3.3 related with PCA.
pznueva <- prcomp(USAFSurvey_preproc$data, scale = TRUE, retx = TRUE)
# Table 3:
summary(pznueva)
pznueva
# PCA scores for 3 archetypes:
p3 <- predict(pznueva,archetypes::parameters(a3))
# PCA scores for 7 archetypes:
p7 <- predict(pznueva,archetypes::parameters(a7))
# Representing the scores:
# Figure 4 (a):
xyplotPCArchetypes(p3[,1:2], pznueva$x[,1:2], data.col = gray(0.7), atypes.col = 1, atypes.pch = 15)
# Figure 4 (b):
xyplotPCArchetypes(p7[,1:2], pznueva$x[,1:2], data.col = gray(0.7), atypes.col = 1, atypes.pch = 15)

# Percentiles for 7 archetypes (table 5):
Fn <- ecdf(USAFSurvey_preproc$data)
round(Fn(archetypes::parameters(a7)) * 100)

# Which are the nearest individuals to archetypes?:
# Example for three archetypes:
ras <- rbind(archetypes::parameters(a3), USAFSurvey_preproc$data)
dras <- dist(ras, method = "euclidean", diag = FALSE, upper = TRUE, p = 2)
mdras <- as.matrix(dras)
diag(mdras) = 1e+11
numArch <- 3
sapply(seq(length=numArch), nearestToArchetypes, numArch, mdras)

# In addition, we can turn the standardized values to the original variables.
p <- archetypes::parameters(a7)
m <- sapply(USAFSurvey_inch, mean)
s <- sapply(USAFSurvey_inch, sd)
d <- p
for(i in 1:6){
  d[,i] = p[,i] * s[i] + m[i]
}
Table 7:
t(d)

archetypoids  Finding archetypoids

Description
Archetypoid algorithm. It is based on the PAM clustering algorithm. It is made up of two phases (a BUILD phase and a SWAP phase). In the BUILD phase, an initial set of archetypoids is determined. Unlike PAM, this collection is not derived in a stepwise format. Instead, it is suggested you choose the set made up of the nearest individuals returned by the archetypes function of the archetypes R package (Eugster et al. (2009)). This set can be defined in three different ways, see next section arguments. The goal of the SWAP step is the same as that of the SWAP step of PAM, but changing the objective function. The initial vector of archetypoids is attempted to be improved. This is done by exchanging selected individuals for unselected individuals and by checking whether these replacements reduce the objective function of the archetypoid analysis problem.

All details are given in Vinue et al. (2015).

Usage
archetypoids(numArchoid, data, huge=200, step, init, ArchObj, nearest="cand_ns", sequ, aux)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numArchoid</td>
<td>Number of archetypoids (archetypal observations).</td>
</tr>
<tr>
<td>data</td>
<td>Data matrix. Each row corresponds to an observation and each column corre-</td>
</tr>
<tr>
<td></td>
<td>sponds to an anthropometric variable. All variables are numeric.</td>
</tr>
<tr>
<td>huge</td>
<td>This is a penalization added to solve the convex least squares problems reg-</td>
</tr>
<tr>
<td></td>
<td>arding the minimization problem to estimate archetypoids, see Eugster et al.</td>
</tr>
<tr>
<td></td>
<td>(2009). Default value is 200.</td>
</tr>
<tr>
<td>step</td>
<td>Logical value. If TRUE, the archetypoid algorithm is executed repeatedly</td>
</tr>
<tr>
<td></td>
<td>within stepArchetypoids. Therefore, this function requires the next argu-</td>
</tr>
<tr>
<td></td>
<td>ment init (but neither the ArchObj nor the nearest arguments) that specifies the initial</td>
</tr>
</tbody>
</table>
vector of archetypoids, which has already been computed within `stepArchetypoids`. If FALSE, the archetypoid algorithm is executed once. In this case, the `ArchObj` and nearest arguments are required to compute the initial vector of archetypoids.

`init`  
Initial vector of archetypoids for the BUILD phase of the archetypoid algorithm. It is computed within `stepArchetypoids`. See nearest argument below for an explanation of how this vector is calculated.

`ArchObj`  
The list object returned by the `stepArchetypesRawData` function. This function is a slight modification of the original `stepArchetypes` function of `archetypes` to apply the archetype algorithm to raw data. The `stepArchetypes` function standardizes the data by default and this option is not always desired. This list is needed to compute the nearest individuals to archetypes. Required when `step=FALSE`.

`nearest`  
Initial vector of archetypoids for the BUILD phase of the archetypoid algorithm. Required when `step=FALSE`. This initial vector contain the nearest individuals to the archetypes returned by the `archetypes` function of `archetypes` (In Vinue et al. (2015), archetypes are computed after running the archetype algorithm twenty times). This argument is a string vector with three different possibilities. The first and default option is "cand_ns" and allows us to calculate the nearest individuals by computing the Euclidean distance between the archetypes and the individuals and choosing the nearest. It is used in Epifanio et al. (2013). The second option is "cand_alpha" and allows us to calculate the nearest individuals by consecutively identifying the individual with the maximum value of alpha for each archetype, until the defined number of archetypes is reached. It is used in Eugster (2012). The third and final option is "cand_beta" and allows us to calculate the nearest individuals by identifying the individuals with the maximum beta value for each archetype, i.e. the major contributors in the generation of the archetypes.

`sequ`  
Logical value. It indicates whether a sequence of archetypoids (TRUE) or only a single number of them (FALSE) is computed. It is determined by the number of archetypes computed by means of `stepArchetypesRawData`.

`aux`  
If `sequ=FALSE`, this value is equal to `numArchoid`-1 since for a single number of archetypoids, the list associated with the archetype object only has one element.

**Details**

As mentioned, this algorithm is based on PAM. These types of algorithms aim to find good solutions in a short period of time, although not necessarily the best solution. Otherwise, the global minimum solution may always be obtained using as much time as it would be necessary, but this would be very inefficient computationally.

**Value**

A list with the following elements:

- `cases`: Anthropometric cases (final vector of `numArchoid` archetypoids).
- `rss`: Residual sum of squares corresponding to the final vector of `numArchoid` archetypoids.
archetypoids

archet_ini: Vector of initial archetypoids (cand_ns, cand_alpha or cand_beta).

alphas: Alpha coefficients for the optimal vector of archetypoids.

Note

It may happen that archetypes does not find results for numArchoid archetypes. In this case, it is not possible to calculate the vector of nearest individuals and consequently, the vector of archetypoids. Therefore, this function will return an error message.

Author(s)

Irene Epifanio and Guillermo Vinue

References


See Also

stepArchetypesRawData, archetypes, stepArchetypoids

Examples

# Note: For a sportive example, see www.uv.es/vivigui/softw/more_examples.R

# COCKPIT DESIGN PROBLEM:
# As a toy example, only the first 25 individuals are used.
USAFSurvey_First25 <- USAFSurvey[1:25, ]

# Variable selection:
variabl_sel <- c(48, 40, 39, 33, 34, 36)

# Changing to inches:
USAFSurvey_First25_inch <- USAFSurvey_First25[,variabl_sel] / (10 * 2.54)

# Data preprocessing:
USAFSurvey_preproc <- preprocessing(USAFSurvey_First25_inch, TRUE, 0.95, TRUE)

# For reproducing results, seed for randomness:
# suppressWarnings(RNGversion("3.5.0"))
# set.seed(2010)
# Run archetype algorithm repeatedly from 1 to numArch archetypes:
#This is a toy example. In other situation, choose numArch=10 and numRep=20.
numArch <- 5 ; numRep <- 2
lass <- stepArchetypesRawData(data = USAFSurvey_preproc$data, numArch = 1:numArch,  
numRep = numRep, verbose = FALSE)

#To understand the warning messages, see the vignette of the  
#archetypes package.

#screeplot(lass)

numArchoid <- 3 #number of archetypoids.
res_ns <- archetypoids(numArchoid, USAFSurvey_preproc$data, huge = 200, step = FALSE,  
ArchObj = lass, nearest = "cand_ns", sequ = TRUE)

### array3Dlandm

#### Helper function for the 3D landmarks

**Description**

This is a helper function for obtaining the array with the 3D landmarks of the sample objects

**Usage**

array3Dlandm(numLandm,numIndiv,matLandm)

**Arguments**

- **numLandm** Number of landmarks that represent the 3D body of the individuals.
- **numIndiv** Number of individuals to analyze.
- **matLandm** Matrix with the numLandm landmarks for the numIndiv individuals.

**Value**

Array with the 3D landmarks of the sample objects.

**Author(s)**

Guillermo Vinue

**References**


**See Also**

LloydShapes,HartiganShapes,trimmedLloydShapes
bustSizesStandard

Examples

Examples

landmarksNoNa <- na.exclude(landmarksSampleSpaSurv)
numLandmarks <- (dim(landmarksNoNa)[2]) / 3
landmarksNoNa_First50 <- landmarksNoNa[1:50,]
numIndiv <- dim(landmarksNoNa_First50)[1]

array3D <- array3Dlandm(numLandmarks, numIndiv, landmarksNoNa_First50)

bustSizesStandard  Helper function for defining the bust sizes

Description

This is a helper function for defining the twelve bust sizes (from 74 cm to 131 cm) according to the sizes proposed in the European standard on sizing systems. Size designation of clothes. Part 3: Measurements and intervals.

Usage

bustSizesStandard(bustCirc_4, bustCirc_6)

Arguments

bustCirc_4  Sequence of measurements from 74 to 102 in groups of four.
bustCirc_6  Sequence of measurements from 107 to 131 in groups of six.

Value

A list with the following elements:

bustCirc:  Vector of the twelve bust sizes.
nsizes:  Number of bust sizes (twelve).

Author(s)

Guillermo Vinue

References

CCbiclustAnthropo

See Also
trimowa, hipamAnthropom

Examples
bustSizes <- bustSizesStandard(seq(74, 102, 4), seq(107, 131, 6))

Description
This function is the implementation in R of the algorithm that uses the Cheng and Church biclustering method (from now on, CC) to find size groups (biclusters) and disaccommodated individuals.

Designing lower body garments depends not only on the waist circumference (the principal dimension in this case), but also on other secondary control dimensions (for upper body garments the bust circumference is usually required only). Biclustering identifies groups of observations with a similar pattern in a subset of attributes instead of in the whole of them. Therefore, it seems to be more interesting to use a biclustering algorithm with a set of lower body variables.

In Vinue et al. (2014), the way of proceeding was as follows: first, all the body variables related to the lower body part included in the Spanish anthropometric survey were chosen (there were 36). Second, the data set was divided into twelve segments (classes) using waist circumference values according to the European standard. Part 3: Measurements and intervals. Finally, the CC algorithm was applied to each waist class.

Usage
CCbiclustAnthropo(data, waistVariable, waistCirc, lowerVar, nsizes, nBic, diffRanges, percDisac, dir)

Arguments
data Data matrix. Each row corresponds to an observation, and each column corresponds to a variable. All variables are numeric.
waistVariable Vector containing the waist values of the individuals.
waistCirc data is segmented into twelve waist classes. This vector contains the waist values to define each one of the twelve classes.
lowerVar Lower body dimensions.
nsizes Number of waist sizes.
nBic Maximum number of biclusters to be found in each waist size.
diffRanges List with nsizes elements. Each element is a vector whose extremes indicate the acceptable boundaries for selecting variables with a similar scale. This is needed because CC may be very influenced in case of variables involved in the study are on very different scales.
percDisac Proportion of no accommodated sample.
dir Working directory where to save the results.
Interesting results in terms of apparel design were found: an efficient partition into different biclusters was obtained. All individuals in the same bicluster can wear a garment designed for the particular body dimensions (waist and other variables) which were the most relevant for defining the group. Each group is represented by the median woman. Because the CC algorithm is nonexhaustive, i.e. some rows (and columns) do not belong to any bicluster, this property can be used to fix a proportion of no accommodated sample.

This approach was descriptive and exploratory. It is emphasized that this function cannot be used with `sampleSpanishSurvey`, because this data file does not contain variables related to the lower body part in addition to waist and hip. However, this function is included in the package in the hope that it could be helpful or useful for other researchers.

**Value**

A list with the following elements:

- **res**: List with `nsizes` elements. Each element contains the biclustering results for each waist segment.
- **dims**: List with `nsizes` elements. Each element contains the number of variables with a similar scale in each waist segment.
- **delta**: List with `nsizes` elements. Each element contains the delta parameter of the CC algorithm for each waist segment.
- **disac**: List with `nsizes` elements. Each element contains the number of women who not belong to any bicluster for each waist segment.
- **mat**: List with `nsizes` elements. Each element contains the matrix showing which rows belong to each bicluster for each waist segment. This matrix allow us to know whether there are rows that belong to more than one bicluster, that is to say, whether there are overlapping biclusters. This is very important in our application because each individual must be assigned to a single size. See the **Note** section.
- **tab_acc**: List with `nsizes` elements. Each element is a list with four elements. The first component indicates how many individuals belong to a single bicluster and how many do not belong to any bicluster. The second component refers to the number of biclusters found in each segment. The third one indicates the number of women that belong to each waist segment. The fourth one coincides with the **disac** element.
- **ColBics**: List with `nsizes` elements. Each element contains the variables that belong to each bicluster for each waist segment.

**Note**

In order to know whether a row belongs to more than one bicluster, we count the number of 0s in each row of the **mat** matrix returned by this function (see the **Value** section).

In case of there are `res@Number - 1` 0s in each row of **mat**, then each row belongs to only one bicluster. The **mat** matrix indicates with an 1 the rows that make up of the bicluster 1, with a 2 those rows that make up of the bicluster 2 and so on. In addition, it indicates with a 0 the rows that do not belong to any bicluster. Therefore, in order to check overlapping, every row must have a number of 0s equal to the total number of biclusters minus one. This one will indicate that that row belongs
to a single bicluster. Otherwise, every row must have a number of 0s equal to the total number of biclusters. In this case, that row does not belong to any bicluster.

For instance, if we find two biclusters, there should be one or two 0s in each row in case of no overlapping.

**Author(s)**

Guillermo Vinue

**References**


**See Also**

`overlapBiclustersByRows`

**Examples**

```r
## Not run:

#Note: package biclust needed.
#This is an example of using this function with a certain database
#made up of body dimensions related to the lower body part.
data <- dataUser[(waist >= 58) & (waist < 115),] #dataUser is the user database.
rownames(data) <- 1:dim(data)[1]

waist <- data[,"WaistCircumference"]
waist_4 <- seq(58, 86, 4)
waist_6 <- seq(91, 115, 6)
waistCirc <- c(waist_4,waist_6)
nsizes <- length(waistCirc)

#Position of the body variables in the database:
lowerVars <- c(14, 17:25, 27, 28, 65:73, 75, 77:81, seq(100, 116, 2))

nBic <- c(2, 2, 4, rep(5, 7), 3, 3)
```


diffRanges <- list(c(14,20), c(24,30), c(24,30), c(29,35), c(29,35), c(28,35), c(31,38), c(31,38), c(30,37), c(31,38), c(31,38), c(30,37), c(26,33), c(25,32))
percDisac <- 0.01
dir <- "/home/guillermo/"

res_bicl_antropom <- CCbiclustAnthropo(data,waist,waistCirc,lowerVars,
nsizes,nBic,diffRanges,percDisac,dir)
## End(Not run)

### cdfDissWomenPrototypes

CDF for the dissimilarities between women and computed medoids and standard prototypes

**Description**

This function allows us to calculate the Cumulative Distribution Functions for the dissimilarities between all the women and the medoids obtained with the `trimowa` algorithm and for the dissimilarities between all the women and the standard prototypes defined by the European standard. Part 3: Measurements and intervals. In both cases, the dissimilarities have been computed by using the dissimilarity function obtained with `getDistMatrix`.

These types of plots can also be used to identify the expected range of the dissimilarities, that is to say, the values between the 10 and 90th percentiles.

This function was used to obtain the Fig. 11 of Ibanez et al. (2012).

**Usage**

cdfDissWomenPrototypes(min_med,min_med_UNE,main,xlab,ylab,leg,cexLeg,...)

**Arguments**

- `min_med` Vector with the dissimilarities between all the women and the prototypes (medoids) obtained with `trimowa`.
- `min_med_UNE` Vector with the dissimilarities between all the women and the standard prototypes.
- `main` A title for the plot.
- `xlab` A title for the x axis.
- `ylab` A title for the y axis.
- `leg` A character vector to appear in the legend.
- `cexLeg` Character expansion for the legend.
- `...` Further graphical parameters.

**Value**

A device with the desired plot.
Author(s)

Guillermo Vinue

References


See Also

sampleSpanishSurvey, weightsMixtureUB, trimowa, getDistMatrix

Examples

#Loading the data to apply the trimowa algorithm:
dataTrimowa <- sampleSpanishSurvey
dim(dataTrimowa)
# [1] 600 5
numVar <- dim(dataTrimowa)[2]
bust <- dataTrimowa$bust
chest <- dataTrimowa$chest
bustSizes <- bustSizesStandard(seq(74, 102, 4), seq(107, 131, 6))

orness <- 0.7
weightsTrimowa <- weightsMixtureUB(orness, numVar)

numClust <- 3; alpha <- 0.01; niter <- 10; algSteps <- 7
ah <- c(23, 28, 20, 25, 25)

#For reproducing results, seed for randomness:
suppressWarnings(RNGversion("3.5.0"))
set.seed(2014)
numSizes <- bustSizes$nsizes - 1
numSizes <- 2
res_trimowa <- computSizesTrimowa(dataTrimowa, bust, bustSizes$bustCirc, numSizes,
weightsTrimowa, numClust, alpha, niter, algSteps,
ah, FALSE)

#Prototypes obtained with the trimowa algorithm:
prototypes <- anthrCases(res_trimowa, numSizes)
#length(unlist(prototypes)) is (numSizes - 1) * numClust
mechs <- dataTrimowa[unlist(prototypes),]

regr <- lm(chest ~ bust)

#Prototypes defined by the European standard:
hip_UNE <- c(seq(84, 112, 4), seq(117, 132, 5)) ; hip <- rep(hip_UNE,3)
waist_UNE <- c(seq(60, 88, 4), seq(94, 112, 6)) ; waist <- rep(waist_UNE,3)
checkBranchLocalIMO

In the HIPAM algorithm, each (parent) cluster P is investigated to see if it can be divided further into new (child) clusters, or stop (in this case, P would be a terminal node).
In this version of HIPAM, called $HIPAM_{IMOS}$, there are three different stopping criteria: First, if $|P| \leq 2$, then $P$ is a terminal node. If not, the second stopping refers to the INCA (Index Number Clusters Atypical) criterion (Irigoien et al. (2008)): if $INCA_k \leq 0.2$ for all $k$, then $P$ is a terminal node. Finally, the third stopping criteria uses the Mean Split Silhouette. See Vinue et al. (2014) for more details.

The foundation and performance of the HIPAM algorithm is explained in hipamAnthropom.

Usage

\[
\text{checkBranchLocalIMO}(\text{tree}, \text{data}, i, \text{maxsplit}, \text{asw.tol}, \text{local.const}, \text{orness}, \text{type}, \text{ah}, \text{verbose}, ...) \]

Arguments

- **tree**: The clustering tree being defined.
- **data**: Data to be clustered.
- **i**: A specific cluster of the clustering partition in a certain level of the tree.
- **maxsplit**: The maximum number of clusters that any cluster can be divided when searching for the best clustering.
- **asw.tol**: If this value is given, a tolerance or penalty can be introduced ($\text{asw.tol} > 0$ or $\text{asw.tol} < 0$, respectively) in the branch splitting procedure. Default value (0) is maintained. See page 154 of Wit et al. (2004) for more details.
- **local.const**: If this value is given (meaningful values are those between -1 and 1), a proposed partition is accepted only if the associated asw is greater than this constant. Default option for this argument is maintained, that is to say, this value is ignored. See page 154 of Wit et al. (2004) for more details.
- **orness**: Quantity to measure the degree to which the aggregation is like a min or max operation. See weightsMixtureUB and getDistMatrix.
- **type**: Option ‘IMO’ for using $HIPAM_{IMOS}$.
- **ah**: Constants that define the ah slopes of the distance function in getDistMatrix. Given the five variables considered, this vector is c(23,28,20,25,25). This vector would be different according to the variables considered.
- **verbose**: Boolean variable (TRUE or FALSE) to indicate whether to report information on progress.
- **...**: Other arguments that may be supplied.

Value

The new resulting classification tree.

Note

This function belongs to the $HIPAM_{IMOS}$ algorithm and it is not solely used. That is why there is no section of examples in this help page. See hipamAnthropom.
This function was originally created by E. Wit et al., and it is available freely on http://www.math.rug.nl/~ernst/book/smida.html. We have adapted it to incorporate the second stopping criterion related to INCA.

References


See Also

hipamAnthropom

Description

In the HIPAM algorithm, each (parent) cluster P is investigated to see if it can be divided further into new (child) clusters, or stop (in this case, P would be a terminal node).

In this version of HIPAM, called HIPAM_MO, there are two different stopping criteria: First, if |P| leq 2$, then P is a terminal node. If not, the second stopping criteria uses the Mean Split Silhouette. See Vinue et al. (2014) for more details.

The foundation and performance of the HIPAM algorithm is explained in hipamAnthropom.

Usage

checkBranchLocalMO(tree, data, i, maxsplit, asw.tol, local.const, orness, type, ah, verbose,...)
checkBranchLocalMO

Arguments

- **tree**: The clustering tree being defined.
- **data**: Data to be clustered.
- **i**: A specific cluster of the clustering partition in a certain level of the tree.
- **maxsplit**: The maximum number of clusters that any cluster can be divided when searching for the best clustering.
- **asw.tol**: If this value is given, a tolerance or penalty can be introduced (asw.tol > 0 or asw.tol < 0, respectively) in the branch splitting procedure. Default value (0) is maintained. See page 154 of Wit et al. (2004) for more details.
- **local.const**: If this value is given (meaningful values are those between -1 and 1), a proposed partition is accepted only if the associated asw is greater than this constant. Default option for this argument is maintained, that is to say, this value is ignored. See page 154 of Wit et al. (2004) for more details.
- **orness**: Quantity to measure the degree to which the aggregation is like a min or max operation. See `weightsMixtureUB` and `getDistMatrix`.
- **type**: Option 'MO' for using $HIPAM_MO$.
- **ah**: Constants that define the $ah$ slopes of the distance function in `getDistMatrix`. Given the five variables considered, this vector is c(23,28,20,25,25). This vector would be different according to the variables considered.
- **verbose**: Boolean variable (TRUE or FALSE) to indicate whether to report information on progress.
- **...**: Other arguments that may be supplied.

Value

The new resulting classification tree.

Note

This function belongs to the $HIPAM_MO$ algorithm and it is not solely used. That is why there is no section of examples in this help page. See hipamAnthropom.

Author(s)

This function was originally created by E. Wit et al., and it is available freely on http://www.math.rug.nl/~ernst/book/smida.html.

References


computSizesHipamAnthropom

See Also

hipamAnthropom

computSizesHipamAnthropom

Computation of the hipamAnthropom elements for a given number of sizes defined by the EN

Description

This is a helper function for computing the hipamAnthropom elements provided by the hipamAnthropom algorithm for a number of bust sizes defined by the European Normative (EN). Therefore, the hipamAnthropom is used inside this function.

Usage

computSizesHipamAnthropom(dataHip, bust, bustMeasur, nsizes, maxsplit, orness, type, ah, verbose = FALSE)

Arguments

dataHip Data frame.
bust Bust column of the data frame.
bustMeasur Sequence vector of bust measurements (bust sizes) provided by the bustSizesStandard function.
nsizes Number of sizes defined by the European Normative to apply the hipamAnthropom function.
maxsplit, orness, type, ah, verbose Same arguments as those of the hipamAnthropom function.

Value

A list with the same elements as the hipamAnthropom function.

Author(s)

Guillermo Vinue

References


**See Also**

`hipamAnthropom, bustSizesStandard`

**Examples**

```r
dataHipam <- sampleSpanishSurvey
bust <- dataHipam$bust
bustSizes <- bustSizesStandard(seq(74, 102, 4), seq(107, 131, 6))

# For reproducibility, set seed for randomness:
# suppressWarnings(RNGversion("3.5.0"))
# set.seed(2013)
numSizes <- 1
res_hipam <- computSizesHipamAnthropom(dataHipam, bust, bustSizes$bustCirc, numSizes,
                                       maxsplit, orness, type, ah, FALSE)
```

**`computSizesTrimowa`**

*Computation of the trimowa elements for a given number of sizes defined by the EN*

**Description**

This is a helper function for computing the trimowa elements provided by the `trimowa` algorithm for a number of bust sizes defined by the European Normative (EN). Therefore, the `trimowa` is used inside this function. The number of sizes must be bigger than one. For a single size use directly `trimowa`.

**Usage**

```r
computSizesTrimowa(dataTrim, bust, bustMeasur, nsizes, w, numClust, alpha,
                   niter, algSteps, ah, verbose = FALSE)
```

**Arguments**

- `dataTrim`  
  Data frame.
- `bust`  
  Bust column of the data frame.
- `bustMeasur`  
  Sequence vector of bust measurements (bust sizes) provided by the `bustSizesStandard` function.
- `nsizes`  
  Number of sizes defined by the European Normative to apply the `trimowa` function.
- `w, numClust, alpha, niter, algSteps, ah, verbose`  
  Same arguments as those of the `trimowa` function.
Value

A list with the same elements as the `trimowa` function.

Author(s)

Guillermo Vinue

References


See Also

`trimowa`, `bustSizesStandard`

Examples

```r
dataTrimowa <- sampleSpanishSurvey
numVar <- dim(dataTrimowa)[2]
bust <- dataTrimowa$bust
bustSizes <- bustSizesStandard(seq(74, 102, 4), seq(107, 131, 6))
orness <- 0.7
weightsTrimowa <- weightsMixtureUB(orness, numVar)
numClust <- 3 ; alpha <- 0.01 ; niter <- 10 ; algSteps <- 7
ah <- c(23, 28, 20, 25, 25)

#For reproducing results, seed for randomness:
#suppressWarnings(RNGversion("3.5.0"))
#set.seed(2014)
numSizes <- 2
res_trimowa <- computSizesTrimowa(dataTrimowa, bust, bustSizes$bustCirc, numSizes,
                                   weightsTrimowa, numClust, alpha, niter,
                                   algSteps, ah, verbose = FALSE)
```

cube34landm

_Cube of 34 landmarks_

Description

This is a cube made up of 34 landmarks, used as controlled data in the simulation study carried out in the paper referred below.
cube8landm

Usage
cube34landm

Format
An array with one matrix of 34 rows and 3 columns.

Source
Software Rhinoceros.

References

cube8landm  Cube of 8 landmarks

Description
This is a cube made up of 8 landmarks, used as controlled data in the simulation study carried out in the paper referred below.

Usage
cube8landm

Format
An array with one matrix of 8 rows and 3 columns.

Source
Software Rhinoceros.

References
**descrDissTrunks**  
*Description of the dissimilarities between women’s trunks*

**Description**

Unlike archetypes, archetypoids can be computed when features are unavailable. Given a dissimilarity matrix, the classical multidimensional scaling (cMDS) can be applied to obtain a description of the dissimilarities.

In Vinue et al. (2015), the dissimilarity matrix represents the dissimilarities between women’s trunks. After applying the cMDS, the database described here is obtained. Then, the archetypoid algorithm can be applied to this database, see section `examples`.

**Usage**

descrDissTrunks

**Format**

A matrix with 470 rows and 4 columns.

**Source**

Anthropometric survey of the Spanish female population.

**References**


**Examples**

```r
#Database:
#As a toy example, only the first 25 individuals are used.
X <- descrDissTrunks[1:25,]
X <- as.matrix(X)

#Computation of archetypes and archetypoids:
#For reproducing results, seed for randomness:
suppressWarnings(RNGversion("3.5.0"))
set.seed(2010)
#Run archetype algorithm repeatedly from 1 to numArch archetypes:
#This is a toy example. In other situation, choose numArch=10 and numRep=20.
numArch <- 5 ; nrep <- 2
lass <- stepArchetypesRawData(data = X, numArch = 1:numArch, numRep = nrep, verbose = FALSE)
#To understand the warning messages, see the vignette of the
```
#archetypes package.
#screeplot(lass)

numArchoid <- 3
res_archoids_ns <- archetypoids(numArchoid, X, huge = 200, step = FALSE, ArchObj = lass, nearest = "cand.ns", sequ = TRUE)

figures8landm

Figures of 8 landmarks with labelled landmarks

Description

This function allows us to represent the two geometric figures (a cube and a parallelepiped) of 8 landmarks, with the landmark labels. Both appear in the paper Vinue et al. (2016), referred below.

Usage

figures8landm(figure, data)

Arguments

figure A character vector, two values are admitted: if figure="cube", the cube is represented. If figure="paral", the parallelepiped is represented.

data The data with the landmarks of the corresponding figure.

Value

A plot of the cube or the parallelepiped with the landmark labels.

Author(s)

Guillermo Vinue

References


Examples

## Not run:
figures8landm("cube", cube8landm)
figures8landm("paral", parallelep8landm)

## End(Not run)
getBestPamsamIMO

**Description**

The HIPAM algorithm starts with one large cluster and, at each level, a given (parent) cluster is partitioned using PAM.

In this version of HIPAM, called $\text{HIPAM}_{\text{IMO}}$, the number $k$ of (child) clusters is obtained by using the INCA (Index Number Clusters Atypical) criterion (Irigoien et al. (2008)) in the following way: at each node $P$, if there is $k$ such that $\text{INCA}_k > 0.2$, then the $k$ prior to the first largest slope decrease is selected. However, this procedure does not apply either to the top node or to the generation of the new partitions from which the Mean Split Silhouette is calculated. In these cases, even when all $\text{INCA}_k < 0.2$, $k = 3$ is fixed as the number of groups to divide and proceed. See Vinue et al. (2014) for more details.

The foundation and performance of the HIPAM algorithm is explained in hipamAnthropom.

**Usage**

```r
getBestPamsamIMO(data, maxsplit, orness=0.7, type, ah, verbose, ...)
```

**Arguments**

- **data**: Data to be clustered.
- **maxsplit**: The maximum number of clusters that any cluster can be divided when searching for the best clustering.
- **orness**: Quantity to measure the degree to which the aggregation is like a min or max operation. See weightsMixtureUB and getDistMatrix.
- **type**: Option 'IMO' for using $\text{HIPAM}_{\text{IMO}}$.
- **ah**: Constants that define the $ah$ slopes of the distance function in getDistMatrix. Given the five variables considered, this vector is $c(23, 28, 20, 25, 25)$. This vector would be different according to the variables considered.
- **verbose**: Boolean variable (TRUE or FALSE) to indicate whether to report information on progress.
- **...**: Other arguments that may be supplied.

**Value**

A list with the following elements:

- **medoids**: The cluster medoids.
- **clustering**: The clustering partition obtained.
- **asw**: The asw of the clustering.
- **num.of.clusters**: Number of clusters in the final clustering.
- **info**: List that informs about the progress of the clustering algorithm.
getBestPamsamMO

profiles: List that contains the asw and sesw (standard error of the silhouette widths) profiles at each stage of the search.

metric: Dissimilarity used (called 'McCulloch' because the dissimilarity function used is that explained in McCulloch et al. (1998)).

Note

This function belongs to the HIPAM_MO algorithm and it is not solely used. That is why there is no section of examples in this help page. See hipamAnthropom.

Author(s)

This function was originally created by E. Wit et al., and it is available freely on http://www.math.rug.nl/~ernst/book/smida.html. We have adapted it to incorporate the INCA criterion.

References


See Also

hipamAnthropom

---

getBestPamsamMO  Generation of the candidate clustering partition in HIPAM_MO

Description

The HIPAM algorithm starts with one large cluster and, at each level, a given (parent) cluster is partitioned using PAM.

In this version of HIPAM, called HIPAM_MO, the number k of (child) clusters is obtained by maximizing the silhouette width (asw). See Vinue et al. (2014) for more details.

The foundation and performance of the HIPAM algorithm is explained in hipamAnthropom.
getBestPamsamMO

Usage

ggetBestPamsamMO(data, maxsplit, orness=0.7, type, ah, verbose, ...)

Arguments

data Data to be clustered.
maxsplit The maximum number of clusters that any cluster can be divided when searching for the best clustering.
orness Quantity to measure the degree to which the aggregation is like a min or max operation. See weightsMixtureUB and getDistMatrix.
type Option 'MO' for using HIPAM_MO.
ah Constants that define the ah slopes of the distance function in getDistMatrix. Given the five variables considered, this vector is c(23,28,20,25,25). This vector would be different according to the variables considered.
verbose Boolean variable (TRUE or FALSE) to indicate whether to report information on progress.
... Other arguments that may be supplied.

Value

A list with the following elements:

medoids: The cluster medoids.
clustering: The clustering partition obtained.
asw: The asw of the clustering.
num.of.clusters: Number of clusters in the final clustering.
info: List that informs about the progress of the clustering algorithm.
profiles: List that contains the asw and sesw (standard error of the silhouette widths) profiles at each stage of the search.
metric: Dissimilarity used (called 'McCulloch' because the dissimilarity function used is that explained in McCulloch et al. (1998)).

Note

This function belongs to the HIPAM_MO algorithm and it is not solely used. That is why there is no section of examples in this help page. See hipamAnthropom.

Author(s)

This function was originally created by E. Wit et al., and it is available freely on http://www.math.rug.nl/~ernst/book/smida.html.
References


See Also

hipamAnthropom

getDistMatrix

* Dissimilarity matrix between individuals and prototypes

Description

In the definition of a sizing system, a distance function allows us to represent mathematically the idea of garment fit and it is a key element to quantify the misfit between an individual and the prototype.

This function computes the dissimilarity defined in McCulloch et al. (1998), which is used in trimowa and hipamAnthropom. For more details, see also Ibanez et al. (2012) and Vinue et al. (2014).

Usage

getDistMatrix(data,np,nv,w,bl,bh,al,ah,verbose)

Arguments

data Data vector.

np Number of observations in the database.

nv Number of variables in the database.

w Weights for the OWA operator computed by means of weightsMixtureUB.

bl,bh,al,ah Constants required to specify the distance function.

verbose Boolean variable (TRUE or FALSE) to indicate whether to report information on progress.
getDistMatrix

Details

At the computational level, it is assumed that all the bh values are negative, all the bl values are positive and all the al and ah slopes are positive (the sign of al is changed within the function when computing the dissimilarities).

Value

A symmetric np x np matrix of dissimilarities.

Note

This function requires a C code called cast.c. In order to use getDistMatrix outside the package, the dynamic-link library is called by means of the sentence dyn.load("cast.so") (In Windows, it would be dyn.load("cast.dll")).

Author(s)

Juan Domingo

References


See Also

trimowa, hipamAnthropom

Examples

# Data loading:
dataTrimowa <- sampleSpanishSurvey
bust <- dataTrimowa$bust
# First bust class:
data <- dataTrimowa[(bust >= 74) & (bust < 78),]
numVar <- dim(dataTrimowa)[2]

# Weights calculation:
orness <- 0.7
weightsTrimowa <- weightsMixtureUB(orness, numVar)

# Constants required to specify the distance function:
numClust <- 3
bh <- (apply(as.matrix(log(data)),2,range)[2,] - apply(as.matrix(log(data)),2,range)[1,]) / ((numClust-1) * 8)
bl <- -3 * bh
ah <- c(23,28,20,25,25)
al <- 3 * ah

#Data processing.
num.persons <- dim(data)[1]
num.variables <- dim(data)[2]
datam <- as.matrix(data)
datat <- aperm(datam, c(2,1))
dim(datat) <- c(1,num.persons * num.variables)

#Dissimilarity matrix:
D <- getDistMatrix(datat, num.persons, numVar, weightsTrimowa, bl, bh, al, ah, FALSE)

HartiganShapes(array3D,numClust,algSteps=10,niter=10,
stopCr=0.0001,simul,initLl,initials,verbose)

HartiganShapes  Hartigan-Wong k-means for 3D shapes

Description

The basic foundation of k-means is that the sample mean is the value that minimizes the Euclidean distance from each point, to the centroid of the cluster to which it belongs. Two fundamental concepts of the statistical shape analysis are the Procrustes mean and the Procrustes distance. Therefore, by integrating the Procrustes mean and the Procrustes distance we can use k-means in the shape analysis context.

The k-means method has been proposed by several scientists in different forms. In computer science and pattern recognition the k-means algorithm is often termed the Lloyd algorithm (see Lloyd (1982)). However, in many texts, the term k-means algorithm is used for certain similar sequential clustering algorithms. Hartigan and Wong (1979) use the term k-means for an algorithm that searches for the locally optimal k-partition by moving points from one cluster to another.

This function allows us to use the Hartigan-Wong version of k-means adapted to deal with 3D shapes. Note that in the generic name of the k-means algorithm, k refers to the number of clusters to search for. To be more specific in the R code, k is referred to as numClust, see next section arguments.

Usage

HartiganShapes(array3D,numClust,algSteps=10,niter=10,
stopCr=0.0001,simul,initLl,initials,verbose)

Arguments

array3D  Array with the 3D landmarks of the sample objects. Each row corresponds to an observation, and each column corresponds to a dimension (x,y,z).
numClust  Number of clusters.
HartiganShapes

algSteps  Number of steps per initialization. Default value is 10.
niter   Number of random initializations (iterations). Default value is 10.
stopCr Relative stopping criteria. Default value is 0.0001.
simul Logical value. If TRUE, this function is used for a simulation study.
initLl Logical value. If TRUE, see next argument initials. If FALSE, they are new random initial values.
initials If initLl=TRUE, they are the same random initial values used in each iteration of LloydShapes. If initLl=FALSE this argument must be passed simply as an empty vector.
verbose A logical specifying whether to provide descriptive output about the running process.

Details

There have been several attempts to adapt the k-means algorithm in the context of the statistical shape analysis, each one adapting a different version of the k-means algorithm (Amaral et al. (2010), Georgescu (2009)). In Vinué, G. et al. (2014), it is demonstrated that the Lloyd k-means represents a noticeable reduction in the computation involved when the sample size increases, compared with the Hartigan-Wong k-means. We state that Hartigan-Wong should be used in the shape analysis context only for very small samples.

Value

A list with the following elements:

ic1: Optimal clustering.
cases: Anthropometric cases (optimal centers).
vopt: Optimal objective function.

If a simulation study is carried out, the following elements are returned:

ic1: Optimal clustering.
cases: Anthropometric cases (optimal centers).
vopt: Optimal objective function.
compTime: Computational time.
AllRate: Allocation rate.

Note

This function is based on the kmns.m file available from https://github.com/johannesgerer/jburkardt-m/tree/master/asa136

Author(s)

Guillermo Vinué
References


See Also

LloydShapes, trimmedLloydShapes, landmarksSampleSpaSurv, cube8landm, parallelep8landm, cube34landm, parallelep34landm, procGPA, optraShapes, qtranShapes

Examples

```r
#CLUSTERING INDIVIDUALS ACCORDING TO THEIR SHAPE:
landmarksNoNa <- na.exclude(landmarksSampleSpaSurv)
dim(landmarksNoNa)
# [1] 574 198
numLandmarks <- (dim(landmarksNoNa)[2]) / 3
# [1] 66
# As a toy example, only the first 20 individuals are used.
landmarksNoNa_First20 <- landmarksNoNa[1:20,]
(numIndiv <- dim(landmarksNoNa_First20)[1])
# [1] 20

array3D <- array3Dlandm(numLandmarks, numIndiv, landmarksNoNa_First20)
#array3D <- array3D[1:10,,] #to reduce computational times.
#shapes::plotshapes(array3D[,1,])
#calibrate::textxy(array3D[,1,1], array3D[,2,1], labs = 1:numLandmarks, cex = 0.7)

numClust <- 3 ; algSteps <- 1 ; niter <- 1 ; stopCr <- 0.0001
# For reproducing results, seed for randomness:
#suppressWarnings(RNGversion("3.5.0"))
#set.seed(2013)
#resHA <- HartiganShapes(array3D, numClust, algSteps, niter, stopCr, FALSE, FALSE, c(), FALSE)
initials <- list(c(15,10,1))
resHA <- HartiganShapes(array3D, numClust, algSteps, niter, stopCr, FALSE, TRUE, initials, TRUE)

if (!is.null(resHA)) {
  asig <- resHA$ic1  #table(asig) shows the clustering results.
  prototypes <- anthrCases(resHA)
}
# Note: For a simulation study, see www.uv.es/vivigu/softw/more_examples.R
```
Description

The HIerarchical Partitioning Around Medoids clustering method (HIPAM) was originally created to gene clustering (Wit et al. (2004)). The HIPAM algorithm is a divisive hierarchical clustering method based on the PAM algorithm.

This function is a HIPAM algorithm adapted to deal with anthropometric data. To that end, a different dissimilarity function is incorporated. This function is that explained in McCulloch et al. (1998) and it is implemented in `getDistMatrix`. We call it $d_{MO}$. In addition, a different method to obtain a classification tree is also incorporated.

Two HIPAM algorithms are proposed. The first one, called $SHIPAM_{MO}$, is a HIPAM that uses $d_{MO}$. The second one, $SHIPAM_{IMO}$, is a HIPAM algorithm that uses $d_{MO}$ and the INCA (Index Number Clusters Atypical) statistic criterion (Irigoien et al. (2008)) to decide the number of child clusters and as a stopping rule.

See Vinue et al. (2014) for more details.

Usage

`hipamAnthropom(data, asw.tol=0, maxsplit=5, local.const=NULL, orness=0.7, type='MO', ah=c(23,28,20,25,25), verbose,...)`

Arguments

data: Data frame. In our approach, this is each of the subframes originated after segmenting the whole anthropometric Spanish survey into twelve bust segments, according to the European standard on sizing systems. Size designation of clothes. Part 3: Measurements and intervals. Each row corresponds to an observation, and each column corresponds to a variable. All variables are numeric.

asw.tol: If this value is given, a tolerance or penalty can be introduced (asw.tol > 0 or asw.tol < 0, respectively) in the branch splitting procedure. Default value (0) is maintained. See page 154 of Wit et al. (2004) for more details.

maxsplit: The maximum number of clusters that any cluster can be divided into when searching for the best clustering.

local.const: If this value is given (meaningful values are those between -1 and 1), a proposed partition is accepted only if the associated asw is greater than this constant. Default option for this argument is maintained, that is to say, this value is ignored. See page 154 of Wit et al. (2004) for more details.

orness: Quantity to measure the degree to which the aggregation is like a min or max operation. See `weightsMixtureUB` and `getDistMatrix`.

type: Type of HIPAM algorithm to be used. The possible options are 'MO' (for $SHIPAM_{MO}$) and 'IMO' (for $SHIPAM_{IMO}$).
Constants that define the \( ah \) slopes of the distance function in `getDistMatrix`. Given the five variables considered, this vector is \( c(23,28,20,25,25) \). This vector would be different according to the variables considered.

#### verbose

Boolean variable (TRUE or FALSE) to indicate whether to report information on progress.

#### Other arguments that may be supplied to the internal functions of the HIPAM algorithms.

**Details**

The \$HIPAM_MO$ algorithm uses the `getBestPamsamMO` and `checkBranchLocalMO` functions, while the \$HIPAM_IMO$ algorithm uses the `getBestPamsamIMO` and `checkBranchLocalIMO` functions.

For more details of HIPAM, see van der Laan et al. (2003), Wit et al. (2004) and the manual of the smida R package.

**Value**

A list with the following elements:

- **clustering**: Final clustering that corresponds to the last level of the tree.
- **asw**: The asw of the final clustering.
- **n.levels**: Number of levels in the tree.
- **cases**: Anthropometric cases (medoids of all of the clusters in the tree).
- **active**: Activity status of each cluster (FALSE for every cluster of the final partition).
- **development**: Matrix that indicates the ancestors of the final clusters.
- **num.of.clusters**: Number of clusters in the final clustering.
- **metric**: Dissimilarity used (called 'McCulloch' because the dissimilarity function used is that explained in McCulloch et al. (1998)).

**Note**

All the functions related to the HIPAM algorithm were originally created by E. Wit et al., and they are available freely on [http://www.math.rug.nl/~ernst/book/smida.html](http://www.math.rug.nl/~ernst/book/smida.html). In order to develop the \$HIPAM_MO$ and \$HIPAM_IMO$ algorithms, we have used and adapted them.

**Author(s)**

Guillermo Vinue

**References**


See Also

globalPamsamMO, getBestPamsamIMO, checkBranchLocalMO, checkBranchLocalIMO, plotTreeHipamAnthropom.

Examples

```r
#FOR THE SIZES DEFINED BY THE EUROPEAN NORMATIVE:
dataHipam <- sampleSpanishSurvey
bust <- dataHipam$bust
bustSizes <- bustSizesStandard(seq(74, 102, 4), seq(107, 131, 6))

# type <- "IMO"
maxsplit <- 5 ; orness <- 0.7
ah <- c(23, 28, 20, 25, 25)

# For reproducing results, seed for randomness:
# suppressWarnings(RNGversion("3.5.0"))
# set.seed(2013)
numSizes <- 1
res_hipam <- computSizesHipamAnthropom(dataHipam, bust, bustSizes$bustCirc, numSizes, maxsplit, orness, type, ah, FALSE)

fitmodels <- anthrCases(res_hipam, numSizes)
outliers <- trimmOutl(res_hipam, numSizes)

# FOR ANY OTHER DEFINED SIZE:
# For reproducing results, seed for randomness:
# suppressWarnings(RNGversion("3.5.0"))
# set.seed(1900)
rand <- sample(1:600,20)
dataComp <- sampleSpanishSurvey[rand, c(2, 3, 5)]
```
numVar <- dim(dataComp)[2]

type <- "IMO"
maxsplit <- 5; orness <- 0.7
ah <- c(28, 25, 25)

dataMat <- as.matrix(dataComp)
#For reproducing results, seed for randomness:
#suppressWarnings(RNGversion("3.5.0"))
#set.seed(2013)
res_hipam_One <- list(); class(res_hipam_One) <- "hipamAnthropom"
res_hipam_One[[1]] <- hipamAnthropom(dataMat, maxsplit = maxsplit, orness = orness,
                      type = type, ah = ah, verbose = FALSE)

#plotTreeHipamAnthropom(res_hipam_One, main="Proposed Hierarchical PAM Clustering 
")
fitmodels_One <- anthrCases(res_hipam_One, 1)
outliers_One <- trimmOutl(res_hipam_One, 1)

###

landmarksSampleSpaSurv

*Landmarks of the sampled women of the Spanish Survey*

**Description**

The body shape of the women who belong to `sampleSpanishSurvey` is represented by a set of anatomical correspondence points, called landmarks.

This database collects the set of landmarks of each woman.

The landmarks considered were placed in three different ways:

- **Automatic landmarks:** automatically calculated with scanner program algorithms, based on geometrical features of the body.
- **Manual landmarks:** points which are not reflected on the external body geometry; they were located through palpation by expert personnel and identified by a physical marker.
- **Digital landmarks:** detected on the computer screen in the 3D scanned image. They are not robust on the automatic calculation but are easy to detect on the screen.

**Usage**

`landmarksSampleSpaSurv`

**Format**

A data frame with 600 observations and 198 variables (66 landmarks times 3 dimensions).

**Source**

Anthropometric survey of the Spanish female population.
LloydShapes

References


LloydShapes

Lloyd k-means for 3D shapes

Description

The basic foundation of k-means is that the sample mean is the value that minimizes the Euclidean distance from each point, to the centroid of the cluster to which it belongs. Two fundamental concepts of the statistical shape analysis are the Procrustes mean and the Procrustes distance. Therefore, by integrating the Procrustes mean and the Procrustes distance we can use k-means in the shape analysis context.

The k-means method has been proposed by several scientists in different forms. In computer science and pattern recognition the k-means algorithm is often termed the Lloyd algorithm (see Lloyd (1982)).

This function allows us to use the Lloyd version of k-means adapted to deal with 3D shapes. Note that in the generic name of the k-means algorithm, k refers to the number of clusters to search for. To be more specific in the R code, k is referred to as numClust, see next section arguments.

Usage

LloydShapes(array3D,numClust,algSteps=10,niter=10,stopCr=0.0001,simul,verbose)

Arguments

array3D Array with the 3D landmarks of the sample objects. Each row corresponds to an observation, and each column corresponds to a dimension (x,y,z).

numClust Number of clusters.

algSteps Number of steps of the algorithm per initialization. Default value is 10.

niter Number of random initializations (iterations). Default value is 10.

stopCr Relative stopping criteria. Default value is 0.0001.

simul Logical value. If TRUE, this function is used for a simulation study.

verbose A logical specifying whether to provide descriptive output about the running process.
Details

There have been several attempts to adapt the k-means algorithm in the context of the statistical shape analysis, each one adapting a different version of the k-means algorithm (Amaral et al. (2010), Georgescu (2009)). In Vinue et al. (2014), it is demonstrated that the Lloyd k-means represents a noticeable reduction in the computation involved when the sample size increases, compared with the Hartigan-Wong k-means. We state that Hartigan-Wong should be used in the shape analysis context only for very small samples.

Value

A list with the following elements:

- `asig`: Optimal clustering.
- `cases`: Anthropometric cases (optimal centers).
- `vopt`: Optimal objective function.
- `initials`: Random initial values used in each iteration. These values are then used by `HartiganShapes`.

If a simulation study is carried out, the following elements are returned:

- `asig`: Optimal clustering.
- `cases`: Anthropometric cases (optimal centers).
- `vopt`: Optimal objective function.
- `compTime`: Computational time.
- `AllRate`: Allocation rate.
- `initials`: Random initial values used in each iteration. These values are then used by `HartiganShapes`.

Author(s)

Amelia Simo

References


See Also

`HartiganShapes`, `trimmedLloydShapes`, `landmarksSampleSpaSurv`, `cube8landm`, `parallelep8landm`, `cube34landm`, `parallelep34landm`, `procGPA`, `optraShapes`, `qtranShapes`
Examples

```r
# CLUSTERING INDIVIDUALS ACCORDING TO THEIR SHAPE:
landmarksNoNa <- na.exclude(landmarksSampleSpaSurv)
dim(landmarksNoNa)
# [1] 574 198
numLandmarks <- (dim(landmarksNoNa)[2]) / 3
# [1] 66
# As a toy example, only the first 10 individuals are used.
landmarksNoNa_First10 <- landmarksNoNa[1:10, ]
(numIndiv <- dim(landmarksNoNa_First10)[1])
# [1] 10

array3D <- array3Dlandm(numLandmarks, numIndiv, landmarksNoNa_First10)
#shapes::plotshapes(array3D[,1,])
#calibrate::textxy(array3D[,1,], array3D[,2,], labs = 1:numLandmarks, cex = 0.7)

numClust <- 2; algSteps <- 1; niter <- 1; stopCr <- 0.0001
resLL <- LloydShapes(array3D, numClust, algSteps, niter, stopCr, FALSE, FALSE)
asig <- resLL$asig
table(resLL$asig)
prototypes <- anthrCases(resLL)

# Note: For a simulation study, see www.uv.es/vivigui/softw/more_examples.R
```

nearestToArchetypes  

Offers the nearest individual to each archetype can be obtained by simply computing the distance between the archetypes and the individuals and choosing the nearest. This is the procedure to obtain what is called the `cand_ns` vector, see Vinue et al. (2015). It is used within `archetypoids` and `stepArchetypoids`.

Usage

```r
nearestToArchetypes(indivs, numArch, mdras)
```

Arguments

- **indivs**: Vector from 1 to `numArch` of individuals nearest to archetypes.
- **numArch**: Number of archetypes computed.
- **mdras**: Distance matrix between the archetypes and the individuals.

Value

A vector with the nearest individuals to archetypes.
Author(s)
Irene Epifanio

References

See Also
archetypoids, stepArchetypoids, archetypesBoundary

Examples

```r
#COCKPIT DESIGN PROBLEM:
#As a toy example, only the first 25 individuals are used.
USAFSurvey_First25 <- USAFSurvey[1:25, ]

#Variable selection:
variabl_sel <- c(48, 40, 39, 33, 34, 36)

#Changing to inches:
USAFSurvey_First25_inch <- USAFSurvey_First25[,variabl_sel] / (10 * 2.54)

#Data preprocessing:
USAFSurvey_preproc <- preprocessing(USAFSurvey_First25_inch, TRUE, 0.95, TRUE)

res <- archetypesBoundary(USAFSurvey_preproc$data, 5, FALSE, 3)
#To understand the warning messages, see the vignette of the archetypes package.

numArch <- 3
a3 <- archetypes::bestModel(res[[numArch]])
ras <- rbind(archetypes::parameters(a3), USAFSurvey_preproc$data)

mdras <- as.matrix(dras)

sapply(seq(length=numArch), nearestToArchetypes, numArch, mdras)
```

optraShapes
Auxiliary optra subroutine of the Hartigan-Wong k-means for 3D shapes
optraShapes

Description

The Hartigan-Wong version of the k-means algorithm uses two auxiliary algorithms: the optimal transfer stage (optra) and the quick transfer stage (qtran).

This function is the optra subroutine adapted to the shape analysis context. It is used within HartiganShapes. See Hartigan and Wong (1979) for details of the original k-means algorithm and Amaral et al. (2010) for details about its adaptation to shape analysis.

Usage

optraShapes(array3D,n,c,numClust,ic1,ic2,nc,an1,an2,ncp,d,itran,live,indx)

Arguments

array3D Array with the 3D landmarks of the sample objects.
n Number of sample objects.
c Array of centroids.
numClust Number of clusters.
ic1 The cluster to each object belongs.
ic2 This vector is used to remember the cluster which each object is most likely to be transferred to at each step.
nc Number of objects in each cluster.
an1 $an1(l) = nc(l) / (nc(l) - 1), l=1,\ldots,numClust$.
an2 $an2(l) = nc(l) / (nc(l) + 1), l=1,\ldots,numClust$.
cp In the optimal transfer stage, ncp(l) stores the step at which cluster l is last updated, $l=1,\ldots,numClust$.
In the quick transfer stage, ncp(l) stores the step at which cluster l is last updated plus n, $l=1,\ldots,numClust$.
d Vector of distances from each object to every centroid.
itran itran(l) = 1 if cluster l is updated in the quick-transfer stage (0 otherwise), $l=1,\ldots,numClust$.
live Vector that indicates whether a cluster is included in the live set or not.
indx Number of steps since a transfer took place.

Value

A list with the following elements: c,ic1,ic2,nc,an1,an2,ncp,d,itran,live,indx, updated after the optimal transfer stage.

Note

This function belongs to HartiganShapes and it is not solely used. That is why there is no section of examples in this help page.
Note

This function is based on the optra.m file available from https://github.com/johannesgerer/jburkardt-m/tree/master/asa136.

Author(s)

Guillermo Vinue

References


See Also

HartiganShapes

overlapBiclustersByRows

Overlapped biclusters by rows

Description

This function allows us to check which rows belong to more than one bicluster. It is used within the CCbiclustAnthropo function.

Usage

overlapBiclustersByRows(Bic, resBiclust)

Arguments

Bic Bicluster number.
resBiclust An object of class Biclust.
Details

In order to know how this function works, it is necessary to understand the following commands:

- `res.bicl@RowxNumber[,1]` indicates the rows that belong to the bicluster 1, by assigning a TRUE value to the position of those rows inside the original matrix. By using `table(res.bicl@RowxNumber[,1])`, we obtain the number of rows belonging to bicluster 1.
- `1 * res.bicl@RowxNumber[,1]` makes TRUES into 1s.
- `Bic * res.bicl@RowxNumber[,Bic]` makes TRUES into the corresponding value of Bic.

In short, this function puts a 1 in those rows belonging to bicluster 1, a 2 in those ones of bicluster 2, and so on.

The fact that certain columns of the matrix returned by this function have a value different from 0 at the same row, will indicate that that row belong to both biclusters.

This function cannot be used with the data of the package. This function is included in the package in the hope that it could be helpful or useful for other researchers.

Value

A matrix with as many rows as rows of the original matrix, and as many columns as obtained biclusters.

Author(s)

Guillermo Vinue

References


See Also

CCbiclustAnthropo

Examples

```r
## Not run:
#Note: package biclust needed.
#This is an example of using this function with a certain database
#made up of body dimensions related to the lower body part.
data <- dataUser[(waist >= 58) & (waist < 115),] #dataUser is the user database.
rownames(data) <- 1:dim(data)[1]

waist <- data[,“WaistCircumference”]

waist_4 <- seq(58, 86, 4)
waist_6 <- seq(91, 115, 6)
waistCirc <- c(waist_4, waist_6)
```
nsizes <- length(waistCirc)

# Position of the body variables in the database:
lowerVars <- c(14, 17:25, 27, 28, 65:73, 75, 77:81, seq(100, 116, 2))

nBic <- c(2, 2, 4, rep(5, 7), 3, 3)
diffRanges <- list(c(14, 20), c(24, 30), c(24, 30), c(29, 35), c(29, 35),
                   c(28, 35), c(31, 38), c(30, 37), c(26, 33), c(35, 32))
percDisac <- 0.01
dir <- "~/home/guillermo/"

res_bicl_antropom <- CCbiclAnthropo(data, waist, waistCirc, lowerVars,
                                    nsizes, nBic, diffRanges, percDisac, dir)

# For a single size:
size <- 5
res <- res_bicl_antropom[[1]][[size]]
sapply(1:res@Number, overlapBiclustersByRows, res)
## End(Not run)

---

**parallelep34landm**  
*Parallelepiped of 34 landmarks*

**Description**

This is a parallelepiped made up of 34 landmarks, used as controlled data in the simulation study carried out in the paper referred below.

**Usage**

`parallelep34landm`

**Format**

An array with one matrix of 34 rows and 3 columns.

**Source**

Software Rhinoceros.

**References**

**Parallelepiped of 8 landmarks**

**Description**

This is a parallelepiped made up of 8 landmarks, used as controlled data in the simulation study carried out in the paper referred below.

**Usage**

`parallelep8landm`

**Format**

An array with one matrix of 8 rows and 3 columns.

**Source**

Software Rhinoceros.

**References**


---

**percentilsArchetypoid**  *Helper function for computing percentiles of a certain archetypoid*

**Description**

This helper function computes the percentiles of an archetypoid for a given variable. Once these percentile values have been calculated, they can be represented by means of a barplot.

**Usage**

`percentilsArchetypoid(column, indiv, data, digits)`

**Arguments**

- `column`  Numeric variable (column of a data frame).
- `indiv`  A certain archetypoid.
- `data`  Data frame that contains the columns and archetypoids to be analyzed.
- `digits`  Argument of the `round` function (it is a integer indicating the number of decimal places to be used).
Value

Numerical vector with the percentile values of an archetypoid.

Author(s)

Guillermo Vinue

References


See Also

archetypoids

Examples

```r
#COCKPIT DESIGN PROBLEM:
#As a toy example, only the first 25 individuals are used.
USAFSurvey_First25 <- USAFSurvey[1:25, ]
#Variable selection:
variabl_sel <- c(48, 40, 39, 33, 34, 36)
#Changing to inches:
USAFSurvey_First25_inch <- USAFSurvey_First25[,variabl_sel] / (10 * 2.54)

#Data preprocessing:
USAFSurvey_preproc <- preprocessing(USAFSurvey_First25_inch, TRUE, 0.95, TRUE)

#For reproducing results, seed for randomness:
#suppressWarnings(RNGversion("3.5.0"))
#set.seed(2010)

#Run archetype algorithm repeatedly from 1 to numArch archetypes:
#This is a toy example. In other situation, choose numArch=10 and numRep=20.
numArch <- 5 ; numRep <- 2
lass <- stepArchetypesRawData(data = USAFSurvey_preproc$data, numArch = 1:numArch,
                               numRep = numRep, verbose = FALSE)
#To understand the warning messages, see the vignette of the archetype package.
#screeplot(lass)

#Three archetypoids:
numArchoid <- 3
res_ns <- archetypoids(numArchoid, USAFSurvey_preproc$data, huge = 200, step = FALSE,
                        ArchObj = lass, nearest = "cand_ns", sequ = TRUE)

percentilsArchetypoid(1, res_ns$archet[1], USAFSurvey_preproc$data, 0)
```
Description

This function represents the scatter plots of bust circumference against other selected variable (chest, hip, neck to ground or waist) jointly with the prototypes obtained for each bust class provided by either trimowa or hipamAnthropom. In addition, the prototypes defined by the European standard on sizing systems. Size designation of clothes. Part 3: Measurements and intervals can be also displayed.

Usage

plotPrototypes(data, prototypes, nsizes, bustVariable, variable, col, xlim, ylim, main, EN)

Arguments

data: Data frame. It should contain the chest, neck to ground, waist, hip and bust measurements of the individuals. In order to be able to represent them, the name of the columns of the database must be 'chest', 'necktoground', 'waist', 'hip' and 'bust' respectively, see sampleSpanishSurvey. Each row corresponds to an observation, and each column corresponds to a variable. All variables are numeric.

prototypes: Prototypes (medoids) i.e., typical persons within the sample, obtained with trimowa or hipamAnthropom.

nsizes: Number of subsets (classes), into the database is segmented. In our approach, the whole anthropometric Spanish survey is segmented into twelve bust segments, according to the European standard on sizing systems. Size designation of clothes. Part 3: Measurements and intervals.

bustVariable: Bust variable.

variable: Anthropometric variable to be plotted. It can be 'chest', 'necktoground', 'waist' and 'hip'.

col: A specification for the medoids color in each bust class.

xlim: Axis length of the x axis according to the range of the bust variable.

ylim: Axis length of the y axis according to the range of the selected variable among chest, hip, neck to ground and waist.

main: Main title of the plot.

EN: A logical value. If TRUE, the prototypes defined by the European standard for each variable are represented. See section Details for more details.
Details

In order to check the goodness of trimowa, the sizes defined by the prototypes can be compared with those defined by the European standard to sizing system. This standard establishes 12 sizes according to the combinations of the bust, waist and hip measurements and does not fix neither chest nor height standard measurements. We can approximate the chest measurements through a linear regression analysis, taking the bust measurements detailed in the standard as independent variable. Besides, we take as neck to ground measurements for the standard sizing system, the values 132, 136 and 140 cm because those are the most repeated values and they are those which best cover our data set. See Ibanez et al. (2012) for a complete explanation.

Value

A device with the desired plot.

Note

As mentioned, this function is especially defined for the sizes established by the European standard on sizing systems. Part 3: Measurements and intervals. In order to use this function with other standard, this function must be adapted.

Author(s)

Guillermo Vinue

References


See Also

sampleSpanishSurvey, weightsMixtureUB, trimowa, getDistMatrix, trimmedoid, hipamAnthropom

Examples

#TRIMOWA ALGORITHM:
dataTrimowa <- sampleSpanishSurvey
numVar <- dim(dataTrimowa)[2]
bust <- dataTrimowa$bust
bustSizes <- bustSizesStandard(seq(74, 102, 4), seq(107, 131, 6))
orness <- 0.7
weightsTrimowa <- weightsMixtureUB(orness, numVar)
numClust <- 3; alpha <- 0.01; niter <- 10; algSteps <- 7
ah <- c(23, 28, 20, 25, 25)

# For reproducing results, seed for randomness:
# suppressWarnings(RNGversion("3.5.0"))
# set.seed(2014)
numSizes <- 2
res_trimowa <- computSizesTrimowa(dataTrimowa, bust, bustSizes$bustCirc, numSizes,
                                 weightsTrimowa, numClust, alpha, niter, algSteps,
                                 ah, FALSE)

prototypes <- anthrCases(res_trimowa, numSizes)

bustVariable <- "bust"
xlim <- c(72, 132)
color <- c("black", "red", "green", "blue", "cyan", "brown", "gray",
           "deeppink3", "orange", "springgreen4", "khaki3", "steelblue1")

variable <- "chest"
range(dataTrimowa[,variable])
  # [1] 76.7755 135.8580
ylim <- c(70,140)
title <- "Prototypes \n bust vs chest"

plotPrototypes(dataTrimowa, prototypes, numSizes, bustVariable,
               variable, color, xlim, ylim, title, FALSE)
plotPrototypes(dataTrimowa, prototypes, numSizes, bustVariable,
               variable, color, xlim, ylim, title, TRUE)

# For other plots and an example for the hipam algorithm,
# see www.uv.es/vivigui/softw/more_examples.R

plotTreeHipamAnthropom

**HIPAM dendogram**

### Description

This function represents a dendrogram for the clustering results provided by a HIPAM algorithm. It is a small modification of the original plot.tree function of the `smida` R package, available from [http://www.math.rug.nl/~ernst/book/smida.html](http://www.math.rug.nl/~ernst/book/smida.html).

### Usage

```r
plotTreeHipamAnthropom(x, main,...)
```

### Arguments

- `x` The HIPAM object to be plotted.
- `main` Title of the plot.
- `...` Other arguments that may be supplied.
Value

A device with the desired plot.

Note

This function only represents the 'tree' option of the original plot.tree function of smida, because we believe that this option displays better the clustering results provided by HIPAM than the option '2d'.

Author(s)

This function was originally created by E. Wit et al., and it is available freely on http://www.math.rug.nl/~ernst/book/smida.html. We have slightly modified.

References


See Also

hipamAnthropom

Examples

dataHipam <- sampleSpanishSurvey
bust <- dataHipam$bust
bustSizes <- bustSizesStandard(seq(74, 102, 4), seq(107, 131, 6))

  type <- "IMO"
  maxsplit <- 5 ; orness <- 0.7
  ah <- c(23, 28, 20, 25, 25)

  #For reproducing results, seed for randomness:
  #suppressWarnings(RNGversion("3.5.0"))
  #set.seed(2013)
  numSizes <- 1
  res_hipam <- computSizesHipamAnthropom(dataHipam, bust, bustSizes$bustCirc, numSizes,
                                          maxsplit, orness, type, ah, FALSE)

  plotTreeHipamAnthropom(res_hipam[[1]],
                        main=paste("Proposed Hierarchical PAM Clustering 
                        "74-78"))
**plotTrimmOutl**

**Trimmed or outlier observations representation**

**Description**

This function represents the scatter plots of bust circumference against other selected variable (chest, hip, neck to ground or waist) jointly with the trimmed individuals discarded in each bust class provided by `trimowa` or with the outlier individuals provided by `hipamAnthropom`.

**Usage**

```r
plotTrimmOutl(data, trimmOutl, nsizes, bustVariable, variable, col, xlim, ylim, main)
```

**Arguments**

- `data`: Data frame. It should contain the chest, neck to ground, waist, hip and bust measurements of the individuals. In order to be able to represent them, the name of the columns of the database must be 'chest', 'necktoground', 'waist', 'hip' and 'bust' respectively, see `sampleSpanishSurvey`. Each row corresponds to an observation, and each column corresponds to a variable. All variables are numeric.

- `trimmOutl`: Trimmed women (if `trimowa`) or outlier women (if `hipamAnthropom`).

- `nsizes`: Number of subsets (classes), into the database is segmented. In our approach, the whole anthropometric Spanish survey is segmented into twelve bust segments, according to the European standard on sizing systems. Size designation of clothes. Part 3: Measurements and intervals.

- `bustVariable`: Bust variable.

- `variable`: Anthropometric variable to be plotted. It can be 'chest', 'necktoground', 'waist' and 'hip'.

- `col`: A specification for the trimmed or outlier women color in each bust class.

- `xlim`: Axis length of the x axis according to the range of the bust variable.

- `ylim`: Axis length of the y axis according to the range of the selected variable among chest, hip, neck to ground and waist.

- `main`: Title of the plot.

**Value**

A device with the desired plot.

**Author(s)**

Guillermo Vinue
References


See Also

sampleSpanishSurvey, hipamAnthropom, trimowa

Examples

#TRIMOWA ALGORITHM:

dataTrimowa <- sampleSpanishSurvey
numVar <- dim(dataTrimowa)[2]
bust <- dataTrimowa$bust
bustSizes <- bustSizesStandard(seq(74, 102, 4), seq(107, 131, 6))
orness <- 0.7
weightsTrimowa <- weightsMixtureUB(orness, numVar)

numClust <- 3 ; alpha <- 0.01 ; niter <- 10 ; algSteps <- 7
ah <- c(23, 28, 20, 25, 25)

#For reproducing results, seed for randomness:
#suppressWarnings(RNGversion("3.5.0"))
#set.seed(2014)
numSizes <- 2
res_trimowa <- computSizesTrimowa(dataTrimowa, bust, bustSizes$bustCirc, numSizes, weightsTrimowa, numClust, alpha, niter, algSteps, ah, FALSE)

prototypes <- anthrCases(res_trimowa, numSizes)
trimmed <- trimmOutl(res_trimowa, numSizes)

bustVariable <- "bust"
xlim <- c(72, 132)

variable <- "chest"
range(dataTrimowa[,variable])
#[1] 76.7755 135.8580
ylim <- c(70, 140)

plotTrimmOutl(dataTrimowa, trimmed, numSizes, bustVariable, variable, color, xlim, ylim, main)

#For other plots and an example for the hipam algorithm,
preprocessing

#see www.uv.es/vivigui/softw/more_examples.R

---

**preprocessing**  
*Data preprocessing before computing archetypal observations*

**Description**

This function allows us to fix the accommodated data before computing archetypes and archetypes. First, depending on the problem, it is possible to standardize the data or not. Second, it is possible to use the Mahalanobis distance or a depth procedure to select the accommodated subsample of data.

**Usage**

```
preprocessing(data, stand, percAccomm, mahal = TRUE)
```

**Arguments**

- **data**: Raw data. It must be a data frame. Each row corresponds to an observation and each column corresponds to an anthropometric variable. All variables are numeric.
- **stand**: A logical value. If TRUE (FALSE) the data are (not) standardized. This option will depend on the problem.
- **percAccomm**: Percentage of the population to accommodate (value between 0 and 1). When this percentage is equal to 1 all the individuals will be accommodated.
- **mahal**: If percAccom is different from 1, then mahal = TRUE (mahal = FALSE) indicates that the Mahalanobis distance (a depth procedure) will be used to select the accommodated subsample of data.

**Details**

In some cases, the depth procedure has the disadvantage that the desired percentage of accommodation is not under control of the analyst and it could not coincide exactly with percAccomm.

**Value**

A list with the following elements if percAccomm is different from 1:

- **data**: Database after preprocessing, with the 1-percAccomm percentage of individuals removed.
- **indivYes**: Individuals who belong to data.
- **indivNo**: Individuals discarded in the accommodation procedure.

A list with the following elements if percAccomm is equals to 1:

- **data**: Initial database with the same number of observations, which has been standarized depending on the value of stand.
Author(s)
Irene Epifanio and Guillermo Vinue

References

Examples
#As a toy example, only the first 25 individuals are used.
#Variable selection:
variabl_sel <- c(48, 40, 39, 33, 34, 36)
#Changing to inches:
USAFSurvey_inch <- USAFSurvey[1:25, variabl_sel] / (10 * 2.54)

#Data preprocessing:
preproc <- preprocessing(USAFSurvey_inch, TRUE, 0.95, TRUE)
preproc <- preprocessing(USAFSurvey_inch, TRUE, 0.95, FALSE)

projShapes

Description
Helper function for plotting the shapes. It displays the projection on the xy plane of the recorded points and mean shape for a given cluster. To that end, first it is needed to carry out a generalized Procrustes analysis in the cluster to obtain the full Procrustes rotated data.

Usage
projShapes(clust, array3D, asig, prototypes)

Arguments
clust Cluster for which represent its mean shape together with the recorded points.
array3D Array with the 3D landmarks of the sample points. Each row corresponds to an observation, and each column corresponds to a dimension (x,y,z).
asig Clustering optimal results.
prototypes Vector of optimal prototypes.

Value
Numerical vector with the percentile values of an archetypoid.
Author(s)
Guillermo Vinue

References

See Also
LloydShapes, HartiganShapes, trimmedLloydShapes

Examples
```r
landmarksNoNa <- na.exclude(landmarksSampleSpaSurv)
dim(landmarksNoNa)
# [1] 574 198
numLandmarks <- (dim(landmarksNoNa)[2]) / 3
# [1] 66
# As a toy example, only the first 15 individuals are used.
landmarksNoNa_First10 <- landmarksNoNa[1:10, ]
(numIndiv <- dim(landmarksNoNa_First10)[1])
# [1] 10

array3D <- array3Dlandm(numLandmarks, numIndiv, landmarksNoNa_First10)
#shapes::plotshapes(array3D[,1])
#calibrate::textxy(array3D[,1], array3D[,2], labs = 1:numLandmarks, cex = 0.7)

numClust <- 2 ; algSteps <- 1 ; niter <- 1 ; stopCr <- 0.0001
resLL <- LloydShapes(array3D, numClust, algSteps, niter, stopCr, FALSE, FALSE)
clust_kmeansProc <- resLL$asig

prototypes <- anthrCases(resLL)

projShapes(1, array3D, clust_kmeansProc, prototypes)
#legend("topleft", c("Registrated data", "Mean shape"), pch = 1, col = 1:2, text.col = 1:2)
#title("Procrustes registrated data for cluster 1 \nwith its mean shape superimposed",
#       sub = "Plane xy")
```

qtranShapes

Auxiliary *qtran* subroutine of the Hartigan-Wong k-means for 3D shapes

Description
The Hartigan-Wong version of the k-means algorithm uses two auxiliary algorithms: the optimal transfer stage (optra) and the quick transfer stage (qtran).

This function is the *qtran* subroutine adapted to the shape analysis context. It is used within *HartiganShapes*. See Hartigan and Wong (1979) for details of the original k-means algorithm and Amaral et al. (2010) for details about its adaptation to shape analysis.
qtranShapes

Usage

qtranShapes(array3D,n,c,ic1,ic2,nc,an1,an2,ncp,d,itran,indx)

Arguments

array3D  Array with the 3D landmarks of the sample objects.
n  Number of sample objects.
c  Array of centroids.
ic1  The cluster to each object belongs.
ic2  This vector is used to remember the cluster which each object is most likely to be transferred to at each step.
nc  Number of objects in each cluster.
an1  $an1(l) = nc(l) / (nc(l) - 1), l=1,\ldots,\text{numClust}$, where numClust is the number of clusters.
an2  $an2(l) = nc(l) / (nc(l) + 1), l=1,\ldots,\text{numClust}$. 
ncp  In the optimal transfer stage, ncp(l) stores the step at which cluster l is last updated, $l=1,\ldots,\text{numClust}$. In the quick transfer stage, ncp(l) stores the step at which cluster l is last updated plus n, $l=1,\ldots,\text{numClust}$. 
d  Vector of distances from each object to every centroid.
itran  itran(l) = 1 if cluster l is updated in the quick-transfer stage (0 otherwise), $l=1,\ldots,k$.
indx  Number of steps since a transfer took place.

Value

A list with the following elements: c, ic1, ic2, nc, an1, an2, ncp, d, itran, indx, icoun, updated after the optimal transfer stage. Note that icoun counts the steps where a re-allocation took place.

Note

This function belongs to HartiganShapes and it is not solely used. That is why there is no section of examples in this help page.

Note

This function is based on the qtran.m file available from https://github.com/johannesgerer/jburkardt-m/tree/master/asa136.

Author(s)

Guillermo Vinue
References


See Also

HartiganShapes

---

sampleSpanishSurvey  
*Sample database of the Spanish anthropometric survey*

### Description

This a database for academic and training purposes. It is oriented to exemplify the use of trimowa, hipamAnthropom and TDDclust.

It is made up of 600 women selected randomly from the Spanish anthropometric survey and five anthropometric variables: chest circumference, neck to ground length, waist circumference, hip circumference and bust circumference. These variables have been chosen following the recommendations of experts. In addition, they are commonly used in the literature about sizing system design and they appear in the European standard to sizing system.

### Usage

```r
sampleSpanishSurvey
```

### Format

A matrix with 600 rows and 5 columns. Each row corresponds to an observation, and each column corresponds to a variable.

### Source

Anthropometric survey of the Spanish female population.
References


See Also

*trimowa, hipamAnthropom, TDDclust*

---

**screeArchetypal**

Screeplot of archetypal individuals

**Description**

This function allows us to represent in the same plot the screeplot of the archetypes and the both `cand_ns`, `cand_alpha` and `cand_beta` archetypoids.

**Usage**

```r
screeArchetypal(numArch,rss_lass_def,rss_step_ns,rss_step_alpha,rss_step_beta,
                 ylim,main,xlab,ylab,col=c("red","blue","green3"),axis2,seq,leg)
```

**Arguments**

- `numArch` Number of archetypal observations (archetypes and archetypoids).
- `rss_lass_def` Vector of the residual sum of squares (rss) associated with each archetype from 1 to `numArch`.
- `rss_step_ns` Vector of the residual sum of squares (rss) associated with each `cand_ns` archetypoid from 1 to `numArch`. 
**screeArchetypal**

-rss_step_alpha Vector of the residual sum of squares (rss) associated with each cand_alpha archetypoid from 1 to numArch.

-rss_step_beta Vector of the residual sum of squares (rss) associated with each cand_beta archetypoid from 1 to numArch.

ylim The y limits of the plot.

main Title of the plot.

xlab A title for the x axis.

ylab A title for the y axis.

col Color vector for the screeplots of the archetypoids. Default is c("red","blue","green3").

axis2 A logical value. If TRUE, the y axis can be customized to have spaced tick-marks by means of the following argument seq.

seq Vector sequence with the values of the tick-marks to be drawn in the y axis.

leg If TRUE, a legend is shown.

**Value**

A device with the desired plot.

**Author(s)**

Guillermo Vinue

**References**


**See Also**

archetypoids, stepArchetypoids

**Examples**

```r
# Not run:
#COCKPIT DESIGN PROBLEM:
#The following R code allows us to obtain a similar plot regarding Figure 5
#of the paper Vinue et al. (2015).
USAFSurvey_First25 <- USAFSurvey[1:25, ]
```
# Variable selection:
variabl_sel <- c(48, 40, 39, 33, 34, 36)

# Changing to inches:
USAFSurvey_First25_inch <- USAFSurvey_First25[, variabl_sel] / (10 * 2.54)

# Data preprocessing:
USAFSurvey_preproc <- preprocessing(USAFSurvey_First25_inch, TRUE, 0.95, TRUE)

# For reproducing results, seed for randomness:
suppressWarnings(RNGversion("3.5.0"))
set.seed(2010)
numArch <- 2
numRep <- 2
lass <- stepArchetypesRawData(data = USAFSurvey_preproc$data,
numArch = 1:numArch, numRep = numRep,
verbose = FALSE)

# To understand the warning messages, see the vignette of the archetypes package.

rss_lass <- matrix(0, nrow = numArch, ncol = numRep)
for(i in 1:numArch){
  for(j in 1:numRep){
    rss_lass[i,j] <- lass[[i]]$rss
  }
}
(rss_lass_def <- apply(rss_lass, 1, min, na.rm = TRUE))

# Run archetypoids algorithm repeatedly from 1 to numArch archetypes:
for(numArchoid in 1:numArch){
  temp <- stepArchetypoids(numArchoid, nearest = "cand_ns",
USAFSurvey_preproc$data, lass)
  filename <- paste("res_ns", numArchoid, sep = "")
  assign(filename, temp)
  save(list = c(filename), file = paste(filename, ".RData", sep = ""))
}

# Run archetypoids algorithm repeatedly from 1 to numArch archetypes:
for(numArchoid in 1:numArch){
  temp <- stepArchetypoids(numArchoid, nearest = "cand_alpha",
USAFSurvey_preproc$data, lass)
  filename <- paste("res_alpha", numArchoid, sep = "")
  assign(filename, temp)
  save(list = c(filename), file = paste(filename, ".RData", sep = ""))
}

# Run archetypoids algorithm repeatedly from 1 to numArch archetypes:
for(numArchoid in 1:numArch){
  temp <- stepArchetypoids(numArchoid, nearest = "cand_beta",
USAFSurvey_preproc$data, lass)
  filename <- paste("res_beta", numArchoid, sep = "")
  assign(filename, temp)
  save(list = c(filename), file = paste(filename, ".RData", sep = ""))
}
}  

# Numerical and graphical results: 
# Cand_ns:  
for(i in 1:numArch){  
  load(paste("res_ns", i, ".RData", sep = ""))  
}  

rss_step <- c()  
for (i in 1:numArch){  
  rss_step[i] <- get(paste("res_ns", i, sep = ""))[2]  
}  
(rss_step_ns <- as.numeric(rss_step))  

# Cand_alpha:  
for(i in 1:numArch){  
  load(paste("res_alpha", i, ".RData", sep = ""))  
}  

rss_step_which <- c()  
for (i in 1:numArch){  
  rss_step_which[i] <- get(paste("res_alpha", i, sep = ""))[2]  
}  
(rss_step_alpha <- as.numeric(rss_step_which))  

# Cand_beta:  
for(i in 1:numArch){  
  load(paste("res_beta", i, ".RData", sep = ""))  
}  

rss_step_which <- c()  
for (i in 1:numArch){  
  rss_step_which[i] <- get(paste("res_beta", i, sep = ""))[2]  
}  
(rss_step_beta <- as.numeric(rss_step_which))  

forYlim <- c(rss_lass_def, rss_step_ns, rss_step_alpha, rss_step_beta)  
range(forYlim)  
# [1] 0.06387125 0.27395811  

# main <- "Aircraft pilots archetypes and archetypoids"  
xlab <- "Archetypes/Archetypoids"  
ylab <- "RSS"  
screeArchetypal(numArch, rss_lass_def, rss_step_ns, rss_step_alpha, rss_step_beta,  
c(0,0.5), main = ", xlab, ylab, col = c("red","blue","green3"),  
TRUE, seq(0,0.5,0.1), FALSE)  

# rm(res_ns1.RData)  
# rm(res_ns2.RData)  
# rm(res_alpha1.RData)  
# rm(res_alpha2.RData)  
# rm(res_beta1.RData)  
# rm(res_beta2.RData)  

## End(Not run)
shapes3dShapes

3D shapes plot

Description

This function is a slight modification of the original shapes3d function of the shapes R package so that the resulting plot has customized title and axes. Specifically, the changing lines regarding the original function are those related to its argument axes3 when it is fixed to TRUE.

Usage

shapes3dShapes(x, loop=0, type="p", color=2, joinline=c(1:1),
axes3=FALSE, rglopen=TRUE, main=main)

Arguments

x
See shapes3d.

loop
See shapes3d.

type
See shapes3d.

color
See shapes3d.

joinline
See shapes3d.

axes3
See shapes3d.

rglopen
See shapes3d.

main
Allows us to give the plot a title if axes3=TRUE.

Value

A device with the desired plot.

References


See Also

shapes3d
skeletonsArchetypal

Examples

```r
## Not run:
landmarksNoNa <- na.exclude(landmarksSampleSpaSurv)
dim(landmarksNoNa)
# [1] 574 198
numLandmarks <- (dim(landmarksNoNa)[2]) / 3
# [1] 66
# As a toy example, only the first 10 individuals are used.
landmarksNoNa_first10 <- landmarksNoNa[1:10,]
(numIndiv <- dim(landmarksNoNa_first10)[1])
# [1] 10
array3D <- array3Dlandm(numLandmarks, numIndiv, landmarksNoNa_first10)
# shapes::plotshapes(array3D[,1])
# calibrate::textxy(array3D[,1,1], array3D[,2,1], labs = 1:numLandmarks, cex = 0.7)
numClust <- 2 ; algSteps <- 1 ; niter <- 1 ; stopCr <- 0.0001
resLL <- LloydShapes(array3D, numClust, algSteps, niter, stopCr, FALSE, FALSE)
prototypes <- anthrCases(resLL)
shapes3dShapes(prototypes[,1,1], loop = 0, type = "p", color = 2, joinline = c(1:1),
                axes3 = TRUE, rglopen = TRUE, main = "Mean shape cluster 1")
## End(Not run)
```

---

skeletonsArchetypal  
**Skeleton plot of archetypal individuals**

Description

This function represents the skeleton plots of the archetypal observations (archetypes and archetypes) of USAFSurvey.

Usage

```r
skeletonsArchetypal(measuArch, main)
```

Arguments

- `measuArch`  Vector with the measurements of each archetype.
- `main`  The title of the plot.

Value

A device with the desired plot.
Note

This function allows us to reproduce the archetypes of Figure 5 of Epifanio et al. (2013), see archetypesBoundary.

Author(s)

Guillermo Vinue

References


See Also

archetypesBoundary, USAFSurvey

Examples

#List with the measurements of each archetype (Table 7 of Epifanio et al (2013)):
lista_arch <- list()
lista_arch[[1]] <- c(34.18, 25.85, 18.65, 39.66, 35.05, 26.73)
lista_arch[[3]] <- c(35.34, 24.94, 18.79, 36.7, 32.28, 23.41)
lista_arch[[4]] <- c(31.34, 22.27, 16.89, 33.08, 33.08, 25.8)
lista_arch[[5]] <- c(32.33, 25.09, 17.84, 34.46, 29.58, 22.82)
lista_arch[[6]] <- c(29.69, 24.18, 18.22, 38.07, 33.04, 24.56)
lista_arch[[7]] <- c(29.24, 22.97, 14.99, 36.88, 32.28, 24.22)

for(i in 1:length(lista_arch)){
  titlePlot <- paste("Archetype", i, sep = " ")
skeletonsArchetypal(lista_arch[[i]],titlePlot)
}

#Note: For an example for archetypoids, see www.uv.es/vivigui/softw/more_examples.R

stepArchetypesRawData  Archetype algorithm to raw data

Description

This is a slight modification of the original stepArchetypes function of the archetypes R package to apply the archetype algorithm to raw data. The stepArchetypes function standardizes the data by default and this option is not always desired.

Usage

stepArchetypesRawData(data,numArch,numRep=3,verbose=TRUE)
Arguments

data  Data to obtain archetypes.
numArch  Number of archetypes to compute, from 1 to numArch.
numRep  For each numArch, run archetypes numRep times.
verbose  If TRUE, the progress during execution is shown.

Value

A list with numArch elements. Each element is a list of class attribute stepArchetypes with numRep elements.

Author(s)

Guillermo Vinue based on the the original stepArchetypes function of archetypes.

References


See Also

stepArchetypes

Examples

#COCKPIT DESIGN PROBLEM:
#As a toy example, only the first 25 individuals are used.
USAFSurvey_First25 <- USAFSurvey[1:25,]

#Variable selection:
variabl_sel <- c(48, 40, 39, 33, 34, 36)

#Changing to inches:
USAFSurvey_First25_inch <- USAFSurvey_First25[,variabl_sel] / (10 * 2.54)

#Data preprocessing:
USAFSurvey_preproc <- preprocessing(USAFSurvey_First25_inch, TRUE, 0.95, TRUE)

#For reproducing results, seed for randomness:
#suppressWarnings(RNGversion("3.5.0"))
#set.seed(2010)

#Run archetype algorithm repeatedly from 1 to numArch archetypes:
#This is a toy example. In other situation, choose numArch=10 and numRep=20.
numArch <- 5 ; numRep <- 2
lass <- stepArchetypesRawData(data = USAFSurvey_preproc$data, numArch = 1:numArch, numRep = numRep, verbose = FALSE)

#To understand the warning messages, see the vignette of the archetypes package.
Run the archetypoid algorithm several times

Description

Execute the archetypoid algorithm repeatedly. It is inspired by the stepArchetypes function of the archetypes R package.

Usage

stepArchetypes(numArchoid, nearest="cand_ns", data, ArchObj)

Arguments

- **numArchoid**: Number of archetypoids.
- **nearest**: Initial vector of archetypoids for the BUILD phase of the archetypoid algorithm. This initial vector contain the nearest individuals to the archetypes returned by the archetypes function of archetypes (In Vinué et al. (2015), archetypes are computed after running the archetype algorithm twenty times). This argument is a string vector with three different possibilities. The first and default option is "cand_ns" and allows us to calculate the nearest individuals by computing the Euclidean distance between the archetypes and the individuals and choosing the nearest. It is used in Epifanio et al. (2013). The second option is "cand_alpha" and allows us to calculate the nearest individuals by consecutively identifying the individual with the maximum value of alpha for each archetype, until the defined number of archetypes is reached. It is used in Eugster (2012). The third and final option is "cand_beta" and allows us to calculate the nearest individuals by identifying the individuals with the maximum beta value for each archetype, i.e. the major contributors in the generation of the archetypes.
- **data**: Data matrix. Each row corresponds to an observation and each column corresponds to an anthropometric variable. All variables are numeric.
- **ArchObj**: The list object returned by the stepArchetypesRawData function. This function is a slight modification of the original stepArchetypes function of archetypes to apply the archetype algorithm to raw data. The stepArchetypes function standardizes the data by default and this option is not always desired. This list is needed to compute the nearest individuals to archetypes.

Value

A list with the following elements:

- **cases**: Anthropometric cases (final vector of numArchoid archetypoids).
- **rss**: Residual sum of squares corresponding to the final vector of numArchoid archetypoids.
- **archet_ini**: Vector of initial archetypoids (cand_ns, cand_alpha or cand_beta).
- **alphas**: Alpha coefficients for the optimal vector of archetypoids.
Note

It may be happen that archetypes does not find results for k archetypes. In this case, it is not possible to calculate the vector of nearest individuals and consequently, the vector of archetypoids. Therefore, this function will return an error message.

Author(s)

Irene Epifanio and Guillermo Vinue

References


See Also

archetypoids, archetypes, stepArchetypes

Examples

```r
#COCKPIT DESIGN PROBLEM:
#As a toy example, only the first 25 individuals are used.
USAFSurvey_First25 <- USAFSurvey[1:25,]

#Variable selection:
variabl_sel <- c(48, 40, 39, 33, 34, 36)
#Changing to inches:
USAFSurvey_First25_inch <- USAFSurvey_First25[,variabl_sel] / (10 * 2.54)

#Data preprocessing:
USAFSurvey_preproc <- preprocessing(USAFSurvey_First25_inch, TRUE, 0.95, TRUE)

#For reproducing results, seed for randomness:
#suppressWarnings(RNGversion("3.5.0"))
#set.seed(2010)
#Run archetype algorithm repeatedly from 1 to numArch archetypes:
#This is a toy example. In other situation, choose numArch=10 and numRep=20.
numArch <- 2 ; numRep <- 2
lass <- stepArchetypesRawData(data = USAFSurvey_preproc$data, numArch = 1:numArch, numRep = numRep, verbose = FALSE)

#To understand the warning messages, see the vignette of the
#archetypes package.
```
# Run archetypoids algorithm repeatedly from 1 to numArch archetypes:
# for(numArchoid in 1:numArch){
# temp <- stepArchetypoids(numArchoid,nearest="cand_ns",USAFSurvey_preproc$data,lass)
# filename <- paste("res", numArchoid, sep="")
# assign(filename,temp)
# save(list=c(filename),file=paste(filename, ".RData", sep=""))
#} 
temp <- stepArchetypoids(2,nearest="cand_ns",USAFSurvey_preproc$data,lass)

---

**TDDclust**  
Trimmed clustering based on L1 data depth

**Description**

This is the trimmed version of the clustering algorithm based on the L1 depth proposed by Rebecka Jornsten (2004). She segments all the observations in clusters, and assigns to each point z in the data space, the L1 depth value regarding its cluster. A trimmed procedure is incorporated to remove the more extreme individuals of each cluster (those one with the lowest depth values), in line with trimowa.

**Usage**

TDDclust(data,numClust,lambda,Th,niter,T0,simAnn,alpha,data1,verbose=TRUE)

**Arguments**

- `data`  
  Data frame. Each row corresponds to an observation, and each column corresponds to a variable. All variables must be numeric.

- `numClust`  
  Number of clusters.

- `lambda`  
  Tuning parameter that controls the influence the data depth has over the clustering, see Jornsten (2004).

- `Th`  
  Threshold for observations to be relocated, usually set to 0.

- `niter`  
  Number of random initializations (iterations).

- `T0`  
  Simulated annealing parameter. It is the current temperature in the simulated annealing procedure.

- `simAnn`  
  Simulated annealing parameter. It is the decay rate, default 0.9.

- `alpha`  
  Proportion of trimmed sample.

- `data1`  
  The same data frame as `data`, used to incorporate the trimmed observations into the rest of them for the next iteration.

- `verbose`  
  A logical specifying whether to provide descriptive output about the running process. Default TRUE.
Value

A list with the following elements:

- **NN**: Cluster assignment, NN[1,] is the final partition.
- **cases**: Anthropometric cases (the multivariate median cluster representatives).
- **DD**: Depth values of the observations (only if there are trimmed observations).
- **Cost**: Final value of the optimal partition.
- **discarded**: Discarded (trimmed) observations.
- **klBest**: Iteration in which the optimal partition was found.

Author(s)

This function has been defined from the original functions developed by Rebecka Jornsten, which were available freely on http://www.stat.rutgers.edu/home/rebecka/DDcl/. However, the link to this page doesn't currently exist as a result of a website redesign.

References


Examples

# In the interests of simplicity of the computation involved, only 15 points are selected:
dataTDDcl <- sampleSpanishSurvey[1:15, c(2, 3, 5)]
dataTDDcl_aux <- sampleSpanishSurvey[1:15, c(2, 3, 5)]

numClust <- 3 ; alpha <- 0.01 ; lambda <- 0.5 ; niter <- 2
Th <- 0 ; T0 <- 0 ; simAnn <- 0.9

# For reproducing results, seed for randomness:
# suppressWarnings(RNGversion("3.5.0"))
# set.seed(2014)
res_TDDcl <- TDDclust(dataTDDcl, numClust, lambda, Th, niter, T0, simAnn, alpha, dataTDDcl_aux,FALSE)

prototypes <- anthrCases(res_TDDcl)

table(res_TDDcl$NN[1,])
res_TDDcl$Cost
res_TDDcl$klBest

trimmed <- trimmOutl(res_TDDcl)
trimmedLloydShapes

Trimmed Lloyd k-means for 3D shapes

Description

The basic foundation of k-means is that the sample mean is the value that minimizes the Euclidean distance from each point, to the centroid of the cluster to which it belongs. Two fundamental concepts of the statistical shape analysis are the Procrustes mean and the Procrustes distance. Therefore, by integrating the Procrustes mean and the Procrustes distance we can use k-means in the shape analysis context.

The k-means method has been proposed by several scientists in different forms. In computer science and pattern recognition the k-means algorithm is often termed the Lloyd algorithm (see Lloyd (1982)).

This function is proposed to incorporate a modification to LloydShapes in order to make the k-means algorithm robust. Robustness is a property very desirable in a lot of applications. As it is well known, the results of the k-means algorithm can be influenced by outliers and extreme data, or bridging points between clusters. Garcia-Escudero et al. (1999) propose a way of making k-means more robust, which combines the k-means idea with an impartial trimming procedure: a proportion alpha (between 0 and 1) of observations are trimmed (the trimmed observations are self-determined by the data). See also trimmedoid.

Note that in the generic name of the k-means algorithm, k refers to the number of clusters to search for. To be more specific in the R code, k is referred to as numClust, see next section arguments.

Usage

trimmedLloydShapes(array3D,n,alpha,numClust,algSteps=10,niter=10, stopCr=0.0001,verbose)

Arguments

array3D Array with the 3D landmarks of the sample objects. Each row corresponds to an observation, and each column corresponds to a dimension (x,y,z).

n Number of individuals.

alpha Proportion of trimmed sample.

numClust Number of clusters.

algSteps Number of steps per initialization. Default value is 10.

niter Number of random initializations (iterations). Default value is 10.

stopCr Relative stopping criteria. Default value is 0.0001.

verbose A logical specifying whether to provide descriptive output about the running process.
Value

A list with the following elements:

- **asig**: Optimal clustering.
- **cases**: Anthropometric cases (optimal centers).
- **vopt**: Optimal objective function.
- **trimmWomen**: List to save the trimmed individual of each iteration.
- **trimmsIter**: Vector with the number of iterations where the optimum was reached. The last number different from NA refers to the last iteration where the final optimum was reached.
- **bestNstep**: Nstep of the iteration where the optimum has reached.
- **initials**: Random initial values used in each iteration. These values can be used by `HartiganShapes`.
- **discarded**: Discarded (trimmed) observations.

Note

We note that adding a trimmed procedure to the Lloyd algorithm is very direct and easy, while for the Hartigan-Wong algorithm, more modifications of the algorithm are needed, which makes the implementation of its trimmed version difficult.

Author(s)

Amelia Simo

References


See Also

- `LloydShapes`, `trimmedoid`

Examples

```r
# CLUSTERING INDIVIDUALS ACCORDING TO THEIR SHAPE:
landmarksNoNa <- na.exclude(landmarksSampleSpaSurv)

dim(landmarksNoNa) # [1] 574 198
numLandmarks <- (dim(landmarksNoNa)[2]) / 3
# [1] 66
```
#As a toy example, only the first 10 individuals are used.
landmarksNoNa_First10 <- landmarksNoNa[1:10,]
(numIndiv <- dim(landmarksNoNa_First10)[1])
# [1] 10

array3D <- array3Dlandm(numLandmarks, numIndiv, landmarksNoNa_First10)

numClust <- 2; alpha <- 0.01; algSteps <- 1; niter <- 1; stopCr <- 0.0001
# For reproducing results, seed for randomness:
# suppressWarnings(RNGversion("3.5.0"))
# set.seed(2013)
res <- trimmedLloydShapes(array3D, numIndiv, alpha, numClust,
algSteps, niter, stopCr, FALSE)

# Optimal partition and prototypes:
clust <- res$asig
table(clust)
prototypes <- anthrCases(res)

# Trimmed individuals:
trimmed <- trimmOutl(res)

---

trimmedoid

**Trimmed k-medoids algorithm**

### Description

This is the trimmed k-medoids algorithm. It is used within trimowa. It is analogous to k-medoids but a proportion alpha of observations is discarded by the own procedure (the trimmed observations are self-determined by the data). Furthermore, the trimmed k-medoids is analogous to trimmed k-means. An algorithm for computing trimmed k-means can be found in Garcia-Escudero et al. (2003). See Ibanez et al. (2012) for more details. Note that in the generic name of the k-medoids algorithm, k refers to the number of clusters to search for. To be more specific in the R code, k is referred to as numClust, see next section *arguments*.

### Usage

```r
trimmedoid(D, numClust, alpha, niter, algSteps=7, verbose)
```

### Arguments

- **D**: Dissimilarity matrix.
- **numClust**: Number of clusters.
- **alpha**: Proportion of trimmed sample.
- **niter**: Number of random initializations (iterations).
- **algSteps**: Number of steps of the algorithm per initialization. Default value is 7.
- **verbose**: A logical specifying whether to provide descriptive output about the running process.
Value

A list with the following elements:

- `vopt`: The objective value.
- `copt`: The trimmed medoids.
- `asig`: The assignment of each observation (asig=0 indicates trimmed individuals).
- `ch`: The goodness index.
- `Dmod`: Modified data with the non-trimmed women.
- `qq`: Vector with the non-trimmed points.

Author(s)

Irene Epifanio

References


See Also

- `sampleSpanishSurvey`, `weightsMixtureUB`, `getDistMatrix`, `trimowa`, `trimmedLloydShapes`

Examples

```r
# Data loading:
dataTrimowa <- sampleSpanishSurvey
bust <- dataTrimowa$bust

# First bust class:
data <- dataTrimowa[(bust >= 74) & (bust < 78),]
umVar <- dim(dataTrimowa)[2]

# Weights calculation:
orness <- 0.7
weightsTrimowa <- weightsMixtureUB(orness,numVar)

# Constants required to specify the distance function:
numClust <- 3
bh <- (apply(as.matrix(log(data)),2,range)[2,] - apply(as.matrix(log(data)),2,range)[1,]) / ((numClust-1) * 8)
bl <- -3 * bh
ah <- c(23,28,20,25,25)
al <- 3 * ah
```
# Data processing.
um.persons <- dim(data)[1]
um.variables <- dim(data)[2]
datam <- as.matrix(data)
datat <- aperm(datam, c(2,1))
dim(datat) <- c(1, num.persons * num.variables)

# Dissimilarity matrix:
D <- getDistMatrix(datat, num.persons, numVar, weightsTrimowa, bl, bh, al, ah, FALSE)
res_trimm <- trimmedoid(D, numClust, 0.01, 6, 7, FALSE)

trimmOutl

Helper generic function for obtaining the trimmed and outlier observations

Description

The methodologies included in this package which are developed to the clothing design problem take into account that a clothing sizing system is intended to cover only what we could call standard population, leaving out those individuals who are extreme with respect to a set of measurements. For "trimowa", "TDDclust" and "kmeansProcrustes" (which refers to as trimmedLloydShapes in this case) these individuals are called trimmed individuals. For the "hipamAnthropom" methodology these individuals are called outlier individuals.

This auxiliary generic function allows the user to identify the discarded individuals computed by each method in an easy way.

Usage

trimmOutl(resMethod, nsizes)
## S3 method for class 'trimowa'
trimmOutl(resMethod, nsizes)
## S3 method for class 'hipamAnthropom'
trimmOutl(resMethod, nsizes)

Arguments

resMethod This is the object which saves the results obtained by the aforementioned methodologies and which contains the discarded individuals to return.

nsizes Number of bust sizes. This argument is needed for the "trimowa" and "hipamAnthropom" methodologies because they can compute the prototypes for any given number of bust sizes.

Value

A vector of class trimmOutl with the discarded observations.
Author(s)

Guillermo Vinue

References


See Also

*trimowa*, *TDDclust*, *hipamAnthropom*, *LloydShapes*, *HartiganShapes*, *trimmedLloydShapes*

Examples

```r
#CLUSTERING INDIVIDUALS ACCORDING TO THEIR SHAPE:
landmarksNoNa <- na.exclude(landmarksSampleSpaSurv)
dim(landmarksNoNa)
# [1] 574 198
numLandmarks <- (dim(landmarksNoNa)[2]) / 3
# [1] 66
#As a toy example, only the first 10 individuals are used.
landmarksNoNa_First10 <- landmarksNoNa[1:10, ]
(numIndiv <- dim(landmarksNoNa_First10)[1])
# [1] 10
array3D <- array3Dlandm(numLandmarks, numIndiv, landmarksNoNa_First10)

numClust <- 2; alpha <- 0.01; algSteps <- 1; niter <- 1; stopCr <- 0.0001
# For reproducing results, seed for randomness:
# suppressWarnings(RNGversion("3.5.0"))
# set.seed(2013)
res_kmeansProc <- trimmedLloydShapes(array3D, numIndiv, alpha, numClust, algSteps, niter, stopCr, FALSE)

trimmed <- trimmOutl(res_kmeansProc)
```
Description

This is the methodology developed in Ibanez et al. (2012) to define an efficient apparel sizing system based on clustering techniques jointly with OWA operators. In our approach, we apply the trimmed k-medoids algorithm (trimmedoid) to the first twelve bust classes according to the sizes defined in the European standard on sizing systems. Size designation of clothes. Part 3: Measurements and intervals.

Usage

trimowa(data,w,numClust,alpha,niter,algSteps,ah=c(23,28,20,25,25),verbose)

Arguments

data: Data frame. In our approach, this is each of the subframes originated after segmenting the whole anthropometric Spanish survey into twelve bust segments, according to the European standard on sizing systems. Size designation of clothes. Part 3: Measurements and intervals. Each row corresponds to an observation, and each column corresponds to a variable. All variables are numeric.
w: The aggregation weights of the OWA operators. They are computed with the weightsMixtureUB.
numClust: Number of clusters.
alpha: Proportion of trimmed sample.
niter: Number of random initializations (iterations).
algSteps: Number of steps of the algorithm per initialization. Default value is 7.
ah: Constants that define the ah slopes of the distance function in getDistMatrix. Given the five variables considered, this vector is c(23,28,20,25,25). This vector would be different according to the variables considered.
verbose: A logical specifying whether to provide descriptive output about the running process.

Value

A list with the following elements:
cases: Anthropometric cases (medoids of the clusters). They are the prototypes obtained for each bust class.
numTrim: Number of trimmed individuals in each bust class.
numClass: Number of individuals in each bust class.
noTrim: Number of of non-trimmed individuals.
C1,C2,C3,C4: Required constant values to define the distance getDistMatrix (C1 is bh, C2 is bl, C3 is ah and C4 is al).
asig: Vector of the clusters to which each individual belongs.
discarded: Discarded (trimmed) individuals.

Author(s)
Guillermo Vinue

References

See Also
sampleSpanishSurvey, weightsMixtureUB, getDistMatrix, trimmedoid

Examples

```r
#FOR THE SIZES DEFINED BY THE EUROPEAN NORMATIVE:
dataTrimowa <- sampleSpanishSurvey
numVar <- dim(dataTrimowa)[2]
bust <- dataTrimowa$bust
bustSizes <- bustSizesStandard(seq(74, 102, 4), seq(107, 131, 6))

orness <- 0.7
weightsTrimowa <- weightsMixtureUB(orness, numVar)

numClust <- 3; alpha <- 0.01; niter <- 10; algSteps <- 7
ah <- c(23, 28, 20, 25, 25)

#For reproducing results, seed for randomness:
#suppressWarnings(RNGversion("3.5.0"))
#set.seed(2014)
numSizes <- 2
res_trimowa <- computSizesTrimowa(dataTrimowa, bust, bustSizes$bustCirc, numSizes,
weightsTrimowa, numClust, alpha, niter, algSteps, ah, FALSE)
prototypes <- anthrCases(res_trimowa, numSizes)

#FOR ANY OTHER DEFINED SIZE:
#For reproducing results, seed for randomness:
#suppressWarnings(RNGversion("3.5.0"))
#set.seed(1900)
rand <- sample(1:600,20)
dataComp <- sampleSpanishSurvey[rand, c(2, 3, 5)]
numVar <- dim(dataComp)[2]

orness <- 0.7
```

weightsTrimowa <- weightsMixtureUB(orness, numVar)
numClust <- 3 ; alpha <- 0.01 ; niter <- 10 ; algSteps <- 7
ah <- c(28, 25, 25)

# For reproducing results, seed for randomness:
# suppressWarnings(RNGversion("3.5.0"))
# set.seed(2014)
res_trimowa <- trimowa(dataComp, weightsTrimowa, numClust, alpha, niter,
 algSteps, ah, verbose = FALSE)
class(res_trimowa) <- "trimowa"
prototypes <- anthrCases(res_trimowa, 1)

---

**USAFSurvey**  
*USAF 1967 survey*

### Description

This data set comes from the 1967 United States Air Force (USAF) survey. The 1967 USAF survey was conducted during the first three months of 1967 under the direction of the Anthropology Branch of the Aerospace Medical Research Laboratory, located in Ohio. Subjects were measured at 17 Air Force bases across the United States of America. A total of 202 variables (including body dimensions and background variables) were taken on 2420 Air Force personnel between 21 and 50 years of age.

Please find in www.uv.es/vivigui/softw/data_information.zip some files that provide a detailed information about this database. Please note that in this documentation 24 variable names are excluded (Vars 9-11, 28, 76-95).

In Epifanio et al. (2013), the column numbers selected were c(48,40,39,33,32) and correspond to 'Thumb tip reach', 'Buttock-Knee length', 'Popliteal height sitting', 'Sitting height', 'Eye height sitting' and 'Shoulder height sitting'.

### Usage

USAFSurvey

### Format

A matrix with 2420 rows and 202 columns. Each row corresponds to an observation, and each column corresponds to a variable.

### Source

weightsMixtureUB

References


Description

This function calculates the weights of the OWA operators. They can be used to adjust the compromise between the style of garments and the general comfort sensation of wearers. This function is used both in trimowa and hipamAnthropom.

Usage

weightsMixtureUB(orness,numVar)

Arguments

- **orness**: Quantity to measure the degree to which the aggregation is like a min or max operation.
- **numVar**: Number of variables of the database.

Value

Vector with the weights.

Author(s)

Guillermo Ayala

References


See Also

dbinom, getDistMatrix, trimowa, hipamAnthropom
Examples

```r
numVar <- dim(sampleSpanishSurvey)[2]
orness <- 0.7
w <- weightsMixtureUB(orness, numVar)
```

Description

This function is a small modification of the generic `xyplot` function of the `archetypes` R package. It shows the scores for the principal components of all individuals jointly with the scores for the computed archetypes. This function is used to obtain the Figure 4 of the subsection 3.3 of Epifanio et al. (2013).

Value

A device with the desired plot.

Note

There are no usage and arguments sections in this help file because they are the same than those of the page 25 of the reference manual of `archetypes`.

Author(s)

Irene Epifanio

References


See Also

`archetypesBoundary`, `USAFSurvey`

Examples

```r
# First, the USAF 1967 database is read and preprocessed (Zehner et al. (1993)).
# Variable selection:
variabl_sel <- c(48, 40, 39, 33, 34, 36)
# Changing to inches:
USAFSurvey_inch <- USAFSurvey[1:25, variabl_sel] / (10 * 2.54)
# Data preprocessing:
USAFSurvey_preproc <- preprocessing(USAFSurvey_inch, TRUE, 0.95, TRUE)
```
#Procedure and results shown in section 2.2.2 and section 3.1:
#For reproducing results, seed for randomness:
#suppressWarnings(RNGversion("3.5.0"))
#set.seed(2010)
res <- archetypesBoundary(USAFSurvey_preproc$data, 15, FALSE, 3)
#To understand the warning messages, see the vignette of the
#archetypes package.

a3 <- archetypes::bestModel(res[[3]])
a7 <- archetypes::bestModel(res[[7]])

pznueva <- prcomp(USAFSurvey_preproc$data, scale = TRUE, retx = TRUE)
#PCA scores for 3 archetypes:
p3 <- predict(pznueva, archetypes::parameters(a3))
#PCA scores for 7 archetypes:
p7 <- predict(pznueva, archetypes::parameters(a7))
#Representing the scores:
#Figure 4 (a):
xyplotPCArchetypes(p3[,1:2], pznueva$x[,1:2], data.col = gray(0.7),
atypes.col = 1, atypes.pch = 15)
#Figure 4 (b):
xyplotPCArchetypes(p7[,1:2], pznueva$x[,1:2], data.col = gray(0.7),
atypes.col = 1, atypes.pch = 15)
Index

* **ANTHROP**
  * Anthropometry-package, 3
* **array**
  * archetypesBoundary, 6
  * archetypoids, 9
  * checkBranchLocalIMO, 19
  * checkBranchLocalMO, 21
  * figures8landm, 28
  * getBestPamsamIMO, 29
  * getBestPamsamMO, 30
  * getDistMatrix, 32
  * HartiganShapes, 34
  * hipamAnthropom, 37
  * LloydShapes, 41
  * nearestToArchetypes, 43
  * optraShapes, 44
  * plotTreeHipamAnthropom, 53
  * preprocessing, 57
  * qtranShapes, 59
  * skeletonsArchetypal, 67
  * stepArchetypesRawData, 68
  * stepArchetypoids, 70
  * TDDclust, 72
  * trimmedLloydShapes, 74
  * trimowa, 80
  * xyplotPCArchetypes, 84
* **datasets**
  * cube34landm, 25
  * cube8landm, 26
  * descrDissTrunks, 27
  * landmarksSampleSpaSurv, 40
  * parallelep34landm, 48
  * parallelep8landm, 49
  * sampleSpanishSurvey, 61
  * USAFSurvey, 82
* **dplot**
  * cdfDissWomenPrototypes, 17
  * plotPrototypes, 51
  * plotTrimmOutl, 55
* **manip**
  * CCbiclustAnthropo, 14
  * overlapBiclustersByRows, 46
* **math**
  * anthrCases, 5
  * array3Dlandm, 12
  * bustSizesStandard, 13
  * computSizesHipamAnthropom, 23
  * computSizesTrimowa, 24
  * percentilsArchetypoid, 49
  * projShapes, 58
  * screeArchetypal, 62
  * trimmedoid, 76
  * trimmOutl, 78
  * weightsMixtureUB, 83
* **multivariate**
  * shapes3dShapes, 66
  * anthrCases, 5
  * Anthropometry-package, 3
  * archetypes, 6, 7, 9–11, 69–71
  * archetypesBoundary, 6, 7, 44, 68, 84
  * archetypoids, 6, 9, 43, 44, 50, 63, 71
  * array3Dlandm, 12
  * bustSizesStandard, 13, 23–25
  * CCbiclustAnthropo, 14, 46, 47
  * cdfDissWomenPrototypes, 17
  * checkBranchLocalIMO, 19, 38, 39
  * checkBranchLocalMO, 21, 38, 39
  * computSizesHipamAnthropom, 23
  * computSizesTrimowa, 24
  * cube34landm, 25, 36, 42
  * cube8landm, 26, 36, 42
  * dbinom, 83
  * descrDissTrunks, 27
  * figures8landm, 28

86
INDEX

getBestPamsamIMO, 29, 38, 39
getBestPamsamMO, 30, 38, 39
getDistMatrix, 17, 18, 20, 22, 29, 31, 32, 33, 37, 38, 52, 77, 80, 81, 83
HartiganShapes, 6, 12, 34, 42, 45, 46, 59–61, 75, 79
hipamAnthropom, 6, 14, 20–24, 29–33, 37, 51, 52, 54–56, 61, 62, 79, 83
landmarksSampleSpaSurv, 36, 40, 42
LloydShapes, 6, 12, 35, 36, 41, 59, 74, 75, 79
nearestToArchetypes, 7, 43
optraShapes, 36, 42, 44
overlapBiclustersByRows, 16, 46
parallelep34landm, 36, 42, 48
parallelep8landm, 36, 42, 49
percentilsArchetypoid, 49
plotPrototypes, 51
plotTreeHipamAnthropom, 39, 53
plotTrimmOutl, 55
preprocessing, 7, 57
procGPA, 36, 42
projShapes, 58
qtranShapes, 36, 42, 59
round, 49
sampleSpanishSurvey, 15, 18, 40, 51, 52, 55, 56, 61, 77, 81
screeArchetypal, 62
shapes3d, 66
shapes3dShapes, 66
skeletonsArchetypal, 67
stepArchetypes, 6, 7, 10, 68–71
stepArchetypesRawData, 7, 10, 11, 68, 70
stepArchetypoids, 6, 9–11, 43, 44, 63, 70
TDDclust, 6, 61, 62, 72, 79
trimmedLloydShapes, 6, 12, 36, 42, 59, 74, 77–79
trimmedoid, 52, 74, 75, 76, 80, 81
trimmOutl, 78
trimowa, 6, 14, 17, 18, 24, 25, 32, 33, 51, 52, 55, 56, 61, 62, 72, 76, 77, 79, 80, 83
USAFSurvey, 6, 7, 67, 68, 82, 84
weightsMixtureUB, 18, 20, 22, 29, 31, 32, 37, 52, 77, 80, 81, 83
xyplotPCArchetypes, 84