Package 'archetypal'

January 27, 2020

Version 1.1.0

Title Finds the Archetypal Analysis of a Data Frame

Description Performs archetypal analysis by using Principal Convex Hull Analysis (PCHA) under a full control of all algorithmic parameters.
It contains a set of functions for determining the initial solution, the optimal algorithmic parameters and the optimal number of archetypes.
Post run tools are also available for the assessment of the derived solution.
Morup, M., Hansen, LK (2012) <doi:10.1016/j.neucom.2011.06.033>.
Hochbaum, DS, Shmoys, DB (1985) <doi:10.1287/moor.10.2.180>.
Eddy, WF (1977) <doi:10.1145/355759.355768>.
Barber, CB, Dobkin, DP, Huhdanpaa, HT (1996) <doi:10.1145/235815.235821>.
Christopoulos, DT (2016) <doi:10.2139/ssrn.3043076>.
Christopoulos, DT (2015) <doi:10.1016/j.jastp.2015.03.009> .
Falk, A. et al. (2018), <doi:10.1093/qje/qjy013> .

Maintainer Demetris Christopoulos <dchristop@econ.uoa.gr>

Depends R (>= 3.1.0)

Imports Matrix, geometry, inflection, doParallel, lpSolve, methods, plot3D

Suggests knitr, rmarkdown

VignetteBuilder knitr

License GPL (>= 2)

Encoding UTF-8

LazyData true

ByteCompile true

NeedsCompilation no

Author Demetris Christopoulos [aut, cre],
David Midgley [ctb],
Sunil Venaik [ctb],
INSEAD Fontainebleau France [fnd, cph]

Repository CRAN

Date/Publication 2020-01-27 10:00:06 UTC

R topics documented:

`archetypal-package` *Finds the Archetypal Analysis of a Data Frame*

Description

Performs archetypal analysis by using Principal Convex Hull Analysis (PCHA) under a full control of all algorithmic parameters. It contains a set of functions for determining the initial solution, the optimal algorithmic parameters and the optimal number of archetypes. Post run tools are also available for the assessment of the derived solution.

Compute Archetypal Analysis (AA)

The main function is `archetypal` which is a variant of PCHA algorithm, see [1], [2], suitable for R language. It provides control to the entire set of involved parameters and has two main options:

1. initialrows = NULL, then a method from "projected_convexhull", "convexhull", "partitioned_convexhul", "furthestsum", "outmost", "random" is used
2. initialrows = (a vector of kappas rows), then given rows form the initial solution for AA

This is the main function of the package, but extensive trials has shown that:

- AA may be very difficult to run if a random initial solution has been chosen

- for the same data set the final Sum of Squared Errors (SSE) may be much smaller if initial solution is close to the final one
- even the quality of AA done is affected from the starting point

This is the reason why we have developed a whole set of methods for choosing initial solution for the PCHA algorithm.

Find a time efficient initial approximation for AA

There are three functions that work with the Convex Hull (CH) of data set.

1. `find_outmost_convexhull_points` computes the CH of all points
2. `find_outmost_projected_convexhull_points` computes the CH for all possible combinations of variables taken by `npr` (default=2)
3. `find_outmost_partitioned_convexhull_points` makes `np` partitions of data frame (defualt=10), then computes CH for each partition and finally gives the CH of overall union

The most simple method for estimating an initial solution is `find_outmost_points` where we just compute the outermost points, i.e. those that are the most frequent outermost for all available points.

The default method "FurthestSum" (FS) of PCHA (see [1], [2]) is used by `find_furthestsum_points` which applies FS for `nfurthest` times (default=10) and then finds the most frequent points.

Of course "random" method is available for comparison reasons and that gives a random set of kappas points as initial solution.

All methods give the number of rows for the input data frame as integers. Attention needed if your data frame has row names which are integers but not identical to `1:dim(df)[1]`.

Find the optimal number of archetypes

For that task `find_optimal_kappas` is available which runs for each kappas from 1 to maxkappas (default=15) ntrials (default=10) times AA, stores SSE, VarianceExplained from each run and then computes knee or elbow point by using UIK method, see [3].

Determining the optimal updating parameters

Extensive trials have shown us that choosing the proper values for algorithmic updating parameters (`muAup,muAdown,muBup,muBdown`) can speed up remarkably the process. That is the task of `find_pcha_optimal_parameters` which conducts a grid search with different values of these parameters and returns the values which minimize the SSE after a fixed number of iterations (`testing_iters`, default=10).

Evaluate the quality of Archetypal Analysis

By using function `check_Bmatrix` we can evaluate the overall quality of applied method and algorithm. Quality can be considered high:

1. if every archetype is being created by a small number of data points
2. if relevant weights are not numerically insignificant

Of course we must take into account the SSE and VarianceExplained, but if we have to compare two solutions with similar termination status, then we must choose that of the simplest B matrix form.

Resampling

The package includes a function for resampling (grouped_resample) which may be used for standard bootstrapping or for subsampling. This function allows samples to be drawn with or without replacement, by groups and with or without Dirichlet weights. This provides a variety of options for researchers who wish to correct sample biases, estimate empirical confidence intervals, and/or subsample large data sets.

Post-run tools

Except from check_Bmatrix there exist next functions for checking the convergence process itself and for examining the local neighborhoud of archetypes:

1. The function study_AAconvergence analyzes the history of iterations done and produces a multi-panel plot showing the steps and quality of the convergence to the final archetypes.
2. By setting the desired number npoints as argument in function find_closer_points we can then find the data points that are in the local neighborhood of each archetype. This allows us to study the properties of the solution or manually choose an initial approximation to search for a better fit.

Note

Bug reports and feature requests can be sent to <dchristop@econ.uoa.gr> or <dem.christop@gmail.com>.

Author(s)

Maintainer: Demetris Christopoulos <dchristop@econ.uoa.gr>

Other contributors:

- David Midgley <david.midgley@insead.edu> [contributor]
- Sunil Venaik <s.venaik@business.uq.edu.au> [contributor]
- INSEAD Fontainebleau France [funder, copyright holder]

References

[1] M Morup and LK Hansen, "Archetypal analysis for machine learning and data mining", Neurocomputing (Elsevier, 2012). https://doi.org/10.1016/j.neucom.2011.06.033.

[2] Source: http://www.mortenmorup.dk/index_files/Page327.htm , last accessed 2019-06-07

[3] Christopoulos, Demetris T., Introducing Unit Invariant Knee (UIK) As an Objective Choice for Elbow Point in Multivariate Data Analysis Techniques (March 1, 2016). Available at SSRN: https://ssrn.com/abstract=3043076 or http://dx.doi.org/10.2139/ssrn.3043076

See Also

archetypal

`AbsoluteTemperature` *Global Absolute Temperature data set for Northern Hemisphere 1969-2013*

Description

It is a subset from the data set which was used for publication [1], i.e. the Global Absolute Temperature for Northern Hemisphere (1800-2013) with only complete yearly observations included. Here we have kept the years 1969-2013.

Usage

```
data("AbsoluteTemperature")
```

Format

A data frame with 155862 observations on the following 18 variables.

`Year` an integer vector of observation years from 1969 to 2013

`Jan` numeric vector of monthly average temperature for January

`Feb` numeric vector of monthly average temperature for February

`Mar` numeric vector of monthly average temperature for March

`Apr` numeric vector of monthly average temperature for April

`May` numeric vector of monthly average temperature for May

`Jun` numeric vector of monthly average temperature for June

`Jul` numeric vector of monthly average temperature for July

`Aug` numeric vector of monthly average temperature for August

`Sep` numeric vector of monthly average temperature for September

`Oct` numeric vector of monthly average temperature for October

`Nov` numeric vector of monthly average temperature for November

`Dec` numeric vector of monthly average temperature for December

`long` a numeric vector for the geographical longitude: positive values for eastings

`lat` a numeric vector for the geographical latitude: positive values for northings

`h` a numeric vector for the altitude in metrs

`stid` an integer vector with the station identity number

`z` an integer vector with the relevant climate zone:

- 1, Tropical Zone
- 2, Subtropics
- 3, Temperate zone
- 4, Cold Zone

Details

That data set was the output of the procedure described in [1]. Initial data set was downloaded from [2] at 2014-12-17.

References

[1] Demetris T. Christopoulos. Extraction of the global absolute temperature for Northern Hemisphere using a set of 6190 meteorological stations from 1800 to 2013. Journal of Atmospheric and Solar-Terrestrial Physics, 128:70 - 83, 3 2015. doi:10.1016/j.jastp.2015.03.009

[2] Met Office Hadley Centre observations datasets, station data sets, http:///www.metoffice.gov.uk/hadobs/crutem4/data/station_files/CRUTEM.4.2.0.0.station_files.zip (last visited 17.12.14)

Examples

```
#
#####################################
## Load absolute temperature data set:
#####################################
#
data("AbsoluteTemperature")
df=AbsoluteTemperature
## Find proportions for climate zones
pcs=table(df$z)/dim(df)[1]
## Choose an approximate size of the new sample and compute resample sizes
N=1000
resamplesizes=as.integer(round(N*pcs))
sum(resamplesizes)
## Create the grouping matrix
groupmat=data.frame("Group_ID"=1:4,"Resample_Size"=resamplesizes)
groupmat
## Simple resampling:
resample_simple <- grouped_resample(in_data = df,grp_vector = "z",
grp_matrix = groupmat,replace = FALSE, option = "Simple", rseed = 20191119)
cat(dim(resample_simple),"\n")
## Dirichlet resampling:
resample_dirichlet <- grouped_resample(in_data = df,grp_vector = "z",
grp_matrix = groupmat, replace = FALSE, option = "Dirichlet", rseed = 20191119)
cat(dim(resample_dirichlet),"\n")
#
########################################
## Reproduce the results of 2015 article
########################################
##
data("AbsoluteTemperature")
dh=AbsoluteTemperature
## Create yearly averages for every station
dh$avg = rowMeans(df[,month.abb[1:12]])
head(dh)
## Compute mean average of every year for all Northern Hemisphere
dagg=data.frame(aggregate(avg~Year,dh,function(x){c(mean(x),sd(x))}))
```

```
## Find used stations per year
daggn=aggregate(stid ~ Year,dh,length)
head(daggn)
tail(daggn)
## Combine all in a data frame
dagyears=data.frame(dagg$Year,daggn$stid,dagg$avg[,1],dagg$avg[,2])
colnames(dagyears)=c("Year","Nv","mu","Smu")
head(dagyears)
tail(dagyears)
#
## Compare with Table 7 (Columns: Year, Nv, mu_bar, Smu_bar), page 77 of article
## Extraction of the global absolute temperature for Northern Hemisphere
## using a set of 6190 meteorological stations from 1800 to 2013
## https://doi.org/10.1016/j.jastp.2015.03.009
## and specifically the years 1969--2013
```

`align_archetypes_from_list` *Align archetypes from a list either by the most frequent found or by using a given archetype*

Description

Align archetypes from a list either by the most frequent or by using a given archetype.

Usage

```
align_archetypes_from_list(archs_list, given_arch = NULL,
  varnames = NULL, ndigits = 0, parallel = FALSE,
  nworkers = NULL, verbose = TRUE)
```

Arguments

`archs_list`	The list of archetypes that must be aligned
`given_arch`	If it is not NULL, then `given_arch` will by used as guide for aligning other archetypes of list. Otherwise, a heuristic for finding the most frequent archetype will be used.
`varnames`	The character vector of variable names that must be used. If it is NULL, then the column names of first archetype will be used.
`ndigits`	The number of digits that will be used for truncation.
`parallel`	If it set to TRUE, then parallel processing will be applied.
`nworkers`	The number of logical processors that will be used for parallel computing (usually it is the double of available physical cores).
`verbose`	If it is set to TRUE, then details are printed out

Value

A list with members:

1. arch_guide, the archetype used as guide for aligning others
2. phrases_most, a table with all rounded phrases from archetypes. Frequencies are in decreasing order, so first row indicates the most frequent sequence, if exists. Otherwise we take randomly a case and proceed.
3. archs_aa_output, a data frame with rows all given archetypes
4. archs_aligned, the final list of aligned archetypes

References

This function is a modification of "align_arc" function from package "ParetoTI", see https://github.com/vitkl/ParetoTI and https://github.com/vitkl/ParetoTI/blob/master/R/align_arc.R

Examples

```
data("wd2") #2D demo
df = wd2
# Define 4 archetypes found for it
dalist = list(c(2.172991,3.200754,5.384013,2.579770,4.860343,3.085111),
              c(5.430821,3.128493,2.043495,3.146342,4.781851,2.710885),
              c(5.430752,2.043403,3.128520,3.146252,2.710979,4.781880),
              c(2.043854,5.430890,3.127183,2.710522,3.146432,4.780432))
archslist = lapply(dalist, function(x){matrix(x,ncol=2)}) #not aligned
# Run aligner
yy = align_archetypes_from_list(archs_list = archslist,
                                given_arch = archslist[[1]])
yy$arch_guide
aligned_archs = yy$archs_aligned
aligned_archs #observe that they are comparable now
```

`archetypal` *archetypal: Finds the archetypal analysis of a data frame by using a variant of the PCHA algorithm*

Description

Performs archetypal analysis by using Principal Convex Hull Analysis (PCHA) under a full control of all algorithmic parameters.

Usage

```
archetypal(df, kappas, initialrows = NULL,
  method = "projected_convexhull", nprojected = 2, npartition = 10,
  nfurthest = 10, maxiter = 2000, conv_crit = 1e-06,
  var_crit = 0.9999, verbose = TRUE, rseed = NULL, aupdate1 = 25,
  aupdate2 = 10, bupdate = 10, muAup = 1.2, muAdown = 0.5,
  muBup = 1.2, muBdown = 0.5, SSE_A_conv = 1e-09,
  SSE_B_conv = 1e-09, save_history = FALSE, nworkers = NULL)
```

Arguments

df	The data frame with dimensions n x d
kappas	The number of archetypes
initialrows	The initial set of rows from data frame that will be used for starting algorithm
method	The method that will be used for computing initial approximation: 1. projected_convexhull, see find_outmost_projected_convexhull_points 2. convexhull, see find_outmost_convexhull_points 3. partitioned_convexhull, see find_outmost_partitioned_convexhull_points 4. furthestsum, see find_furthestsum_points 5. outmost, see find_outmost_points 6. random, a random set of kappas points will be used
nprojected	The dimension of the projected subspace for find_outmost_projected_convexhull_points
npartition	The number of partitions for find_outmost_partitioned_convexhull_points
nfurthest	The number of times that FurthestSum algorithm will be applied by find_furthestsum_points
maxiter	The maximum number of iterations for main algorithm application
conv_crit	The SSE convergence criterion of termination: iterate until |dSSE|/SSE<conv_crit
var_crit	The Variance Explained (VarExpl) convergence criterion of termination: iterate until VarExpl<var_crit
verbose	If it is set to TRUE, then both initialization and iteration details are printed out
rseed	The random seed that will be used for setting initial A matrix. Useful for reproducible results.
aupdate1	The number of initial applications of Aupdate for improving the initially randomly selected A matrix
aupdate2	The number of Aupdate applications in main iteration
bupdate	The number of Bupdate applications in main iteration
muAup	The factor (>1) by which muA is multiplied when it holds SSE<=SSE_old(1+SSE_A_conv)
muAdown	The factor (<1) by which muA is multiplied when it holds SSE>SSE_old(1+SSE_A_conv)
muBup	The factor (>1) by which muB is multiplied when it holds SSE<=SSE_old(1+SSE_B_conv)
muBdown	The factor (<1) by which muB is multiplied when it holds SSE>SSE_old(1+SSE_B_conv)
SSE_A_conv	The convergence value used in SSE<=SSE_old(1+SSE_A_conv). Warning: there exists a Matlab crash sometimes after setting this to 1E-16 or lower

SSE_B_conv	The convergence value used in SSE<=SSE_old(1+SSE_A_conv). Warning: there exists a Matlab crash sometimes after setting this to 1E-16 or lower
save_history	If set TRUE, then iteration history is being saved for further use
nworkers	The number of logical processors that will be used for parallel computing (usually it is the double of available physical cores). Parallel computation is applied when asked by functions find_furthestsum_points, find_outmost_partitioned_convexhull_points and find_outmost_projected_convexhull_points.

Value

A list with members:

1. BY, the $kappas \times d$ matrix of archetypes found
2. A, the $n \times kappas$ matrix such that Y ~ ABY or Frobenius norm ||Y-ABY|| is minimum
3. B, the $kappas \times n$ matrix such that Y ~ ABY or Frobenius norm ||Y-ABY|| is minimum
4. SSE, the sum of squared error SSE = ||Y-ABY||^2
5. varexpl, the Variance Explained = (SST-SSE)/SST where SST is the total sum of squares for data set matrix
6. initialsolution, the initially used set of rows from data frame in order to start the algorithm
7. freqstable, the frequency table for all found rows, if it is available.
8. iterations, the number of main iterations done by algorithm
9. time, the time in seconds that was spent from entire run
10. converges, if it is TRUE, then convergence was achieved before the end of maximum allowed iterations
11. nAup, the total number of times when it was SSE<=SSE_old(1+SSE_A_conv) in Aupdate processes. Useful for debugging purposes.
12. nAdown, the total number of times when it was SSE>SSE_old(1+SSE_A_conv) in Aupdate processes. Useful for debugging purposes.
13. nBup, the total number of times when it was SSE<=SSE_old(1+SSE_B_conv) in Bupdate processes. Useful for debugging purposes.
14. nBdown, the total number of times when it was SSE>SSE_old(1+SSE_A_conv in Bupdate processes. Useful for debugging purposes.
15. run_results, a list of iteration related details: SSE, varexpl, time, B, BY for all iterations done.

References

[1] M Morup and LK Hansen, "Archetypal analysis for machine learning and data mining", Neurocomputing (Elsevier, 2012). https://doi.org/10.1016/j.neucom.2011.06.033.

[2] Source: http://www.mortenmorup.dk/index_files/Page327.htm , last accessed 2019-06-07

Examples

```
# Create a small 2D data set from 3 corner-points:
p1 = c(1,2);p2 = c(3,5);p3 = c(7,3)
dp = rbind(p1,p2,p3);dp
set.seed(916070)
pts = t(sapply(1:20, function(i,dp){
  cc = runif(3)
  cc = cc/sum(cc)
  colSums(dp*cc)
},dp))
df = data.frame(pts)
colnames(df) = c("x","y")
# Run AA:
aa = archetypal(df = df, kappas  =  3, verbose = FALSE, save_history  =  TRUE)
# Archetypes:
archs = data.frame(aa$BY)
archs
# See main results:
names(aa)
aa[c("SSE","varexpl","iterations","time")]
# See history of iterations:
names(aa$run_results)
```

`check_Bmatrix` *Function which checks B matrix of Archetypal Analysis Y ~ A B Y in order to find the used rows for creating each archetype and the relevant used weights.*

Description

Function which checks B matrix of Archetypal Analysis Y ~ A B Y in order to find the used rows for creating each archetype and the relevant used weights.

Usage

```
check_Bmatrix(B, chvertices = NULL, verbose = TRUE)
```

Arguments

`B`	The $kappas \times n$ matrix such that Y ~ ABY or Frobenius norm ‖Y-ABY‖ is minimum
`chvertices`	The vector of rows which represent the Convex Hull of data frame
`verbose`	If set to TRUE, then results are printed out.

Value

A list with members:

1. used_rows, a list with used rows for creating each archetype
2. used_weights, a list with the relevant weights that have been used
3. leading_rows, the rows for each archetype with greatest weight
4. leading_weights, the weights of leading rows
5. used_on_convexhull, the portion of used rows which lie on Convex Hull (if given)

See Also

archetypal, check_Bmatrix, find_closer_points
& study_AAconvergence

Examples

```
{
# Load data "wd2"
data("wd2")
df = wd2
# Run AA:
aa = archetypal(df = df, kappas = 3, verbose = FALSE)
# Check B matrix:
B = aa$B
yy = check_Bmatrix(B, verbose = TRUE)
yy$used_rows
yy$used_weights
yy$leading_rows
yy$leading_weights
# Check if used rows lie on ConvexHull
ch = chull(df)
yy = check_Bmatrix(B, chvertices = ch, verbose = FALSE)
yy$used_on_convexhull
#

}
```

---

dirichlet_sample *Function which performs Dirichlet sampling*

Description

It uses Dirichlet weights for creating sub-samples of initial data set.

Usage

```
dirichlet_sample(in_data = NULL, sample_size = NULL,
replacement = NULL, rseed = NULL)
```

Arguments

in_data	The initial data frame that must be re-sampled. It must contain: 1. an ID variable 2. the variables of interest 3. a grouping variable
sample_size	An integer for the size of the new sample
replacement	A logical input: TRUE/FALSE if replacement should be used or not, respectively
rseed	The random seed that will be used for setting initial A matrix. Useful for reproducible results

Value

It returns a data frame with exactly the same variables as the initial one, except that group variable has now only the given value from input data frame.

Author(s)

David Midgley

See Also

grouped_resample

Examples

```
## Load absolute temperature data set:
data("AbsoluteTemperature")
df=AbsoluteTemperature
## Find portions for climate zones
pcs=table(df$z)/dim(df)[1]
## Choose the approximate size of the new sample and compute resample sizes
N=1000
resamplesizes=as.integer(round(N*pcs))
sum(resamplesizes)
## Create the grouping matrix
groupmat=data.frame("Group_ID"=1:4,"Resample_Size"=resamplesizes)
groupmat
## Dirichlet resampling:
resample_dirichlet <- grouped_resample(in_data = df,grp_vector = "z",
                      grp_matrix = groupmat,replace = FALSE,
                      option = "Dirichlet", rseed = 20191220)
cat(dim(resample_dirichlet),"\n")
```

`find_closer_points` *Function which finds the data points that are closer to the archetypes during all iterations of the algorithm PCHA*

Description

This function runs the PCHA algorithm and finds the data points that are in the local neighborhood of each archetype. The size of the neighborhood is user defined (`npoints`). This allows us to study the properties of the solution or manually choose an initial approximation to search for a better fit.

Usage

```
find_closer_points(df, kappas, usedata = FALSE, npoints = 2,
                   nworkers = NULL, rseed = NULL,
                   verbose = FALSE, doparallel = FALSE, ...)
```

Arguments

`df`	The data frame with dimensions n x d
`kappas`	The number of archetypes
`usedata`	If it is TRUE, then entire data frame will be used, if `doparallel = TRUE`
`npoints`	The number of closer points to be estimated
`nworkers`	The number of logical processors that will be used, if `doparallel = TRUE`
`rseed`	The random seed that will be used for random generator. Useful for reproducible results.
`verbose`	If it is set to TRUE, then details will be printed, except from `archetypal`
`doparallel`	If it is set to TRUE, then parallel processing will be performed
`...`	Other arguments to be passed to `archetypal` except internally used `save_history = TRUE` and `verbose = FALSE`. This is essential for using optimal parameters found by `find_pcha_optimal_parameters`

Value

A list with members:

1. rows_history, a list with `npoints` rows used that are closer to each archetype for each iteration done by algorithm
2. iter_terminal, iteration after which rows closer to archetypes do not change any more
3. rows_closer, the rows closer to archetypes by means of Euclidean distance and are fixed after `iter_terminal` iteration
4. rows_closer_matrix, a matrix with]codenpoints rows which are closer to each archetype
5. solution_used, the AA output that has been used. Some times useful, especially for big data.

See Also

check_Bmatrix, study_AAconvergence

Examples

```
{
# Load data "wd2"
data("wd2")
yy = find_closer_points(df = wd2, kappas = 3, npoints = 2, nworkers = 2)
yy$rows_history
yy$iter_terminal
yy$rows_closer
yy$rows_closer_matrix
yy$solution_used$BY

}
```

`find_furthestsum_points` *Function which finds the furthest sum points in order to be used as initial solution in archetypal analysis*

Description

Function which finds the furthest sum points in order to be used as initial solution in archetypal analysis.

Usage

```
find_furthestsum_points(df, kappas, nfurthest = 100, nworkers = NULL,
                        sortrows = TRUE, doparallel = TRUE)
```

Arguments

`df`	The data frame with dimensions n x d
`kappas`	The number of archetypes
`nfurthest`	The number of applications for FurthestSum algorithm
`nworkers`	The number of logical processors that will be used. Hint: set it such that `nfurthest` can be an exact multiple of `nworkers`.
`sortrows`	If it is TRUE, then rows will be sorted
`doparallel`	If it is set to TRUE, then parallel processing will be performed for the `nfurthest` applications of algorithm

Value

A list with members:

1. outmost, the first kappas furthest sum points as rows of data frame
2. outmostall, all the furthest sum points that have been found as rows of data frame
3. outmostfrequency, a matrix with frequency and cumulative frequency for furthest sum rows

See Also

FurthestSum

Examples

```
data("wd3") #3D demo
df = wd3
yy = find_furthestsum_points(df, kappas = 4, nfurthest = 10, nworkers = 1)
yy$outmost
yy$outmostall
yy$outmostfrequency
```

---

find_optimal_kappas *Function for finding the optimal number of archetypes*

Description

Function for finding the optimal number of archetypes in order to apply Archetypal Analysis for a data frame.

Usage

```
find_optimal_kappas(df, maxkappas = 15, method = "projected_convexhull",
                    ntrials = 10, nworkers = NULL, ...)
```

Arguments

df	The data frame with dimensions $n \times d$
maxkappas	The maximum number of archetypes for which algorithm will be applied
method	The method that will be used for computing the initial solution
ntrials	The number of times that algorithm will be applied for each kappas
nworkers	The number of logical processors that will be used for parallel computing (usually it is the double of available physical cores)
...	Other arguments to be passed to function archetypal

Details

After having found the SSE for each kappas, UIK method (see [1]) is used for estimating the knee or elbow point as the optimal kappas.

Value

A list with members:

1. all_sse, all available SSE for all kappas and all trials per kappas
2. all_sse1, all available SSE(k)/SSE(1) for all kappas and all trials per kappas
3. bestfit_sse, only the best fit SSE trial for each kappas
4. bestfit_sse1, only the best fit SSE(k)/SSE(1) trial for each kappas
5. all_kappas, the knee point of scree plot for all 4 SSE results
6. d2uik, the UIK for the absolute values of the estimated best fit SSE second derivatives, after using second order forward divided differences approximation
7. optimal_kappas, the knee point from best fit SSE results

References

[1] Christopoulos, Demetris T., Introducing Unit Invariant Knee (UIK) As an Objective Choice for Elbow Point in Multivariate Data Analysis Techniques (March 1, 2016). Available at SSRN: http://dx.doi.org/10.2139/ssrn.3043076

See Also

archetypal

Examples

```
{
# Run may take a while depending on your machine ...
# Load data frame "wd2"
data("wd2")
df = wd2
# Run:
t1 = Sys.time()
yy = find_optimal_kappas(df, maxkappas = 10)
t2 = Sys.time();print(t2-t1)
# Results:
names(yy)
# Best fit SSE:
yy$bestfit_sse
# Optimal kappas from UIK method:
yy$optimal_kappas
#

}
```

`find_outmost_convexhull_points` *Function which finds the outermost convex hull points in order to be used as initial solution in archetypal analysis*

Description

Function which finds the outermost convex hull points in order to be used as initial solution in archetypal analysis

Usage

```
find_outmost_convexhull_points(df, kappas)
```

Arguments

`df`	The data frame with dimensions n x d
`kappas`	The number of archetypes

Details

This function uses the `chull` when d=2 (see [1], [2]) and the `convhulln` for d>2 (see [3]) cases.

Value

A list with members:

1. outmost, the first kappas most frequent outermost points as rows of data frame
2. outmostall, all the outermost points that have been found as rows of data frame
3. outmostfrequency, a matrix with frequency and cumulative frequency for outermost rows

References

[1] Eddy, W. F. (1977). A new convex hull algorithm for planar sets. ACM Transactions on Mathematical Software, 3, 398-403. doi: 10.1145/355759.355766.

[2] Eddy, W. F. (1977). Algorithm 523: CONVEX, A new convex hull algorithm for planar sets [Z]. ACM Transactions on Mathematical Software, 3, 411-412. doi: 10.1145/355759.355768.

[3] Barber, C.B., Dobkin, D.P., and Huhdanpaa, H.T., "The Quickhull algorithm for convex hulls" ACM Trans. on Mathematical Software, 22(4):469-483, Dec 1996, http://www.qhull.org

See Also

`find_furthestsum_points`, `find_outmost_projected_convexhull_points`, `find_outmost_partitioned_convexhull_points` & `find_outmost_points`

Examples

```
data("wd2") #2D demo
df = wd2
yy = find_outmost_convexhull_points(df, kappas = 3)
yy$outmost #the rows of 3 outermost points
df[yy$outmost,] #the 3 outermost points
yy$outmostall #all outermost cH rows
yy$outmostfrequency #their frequency
#
###
#
data("wd3") #3D demo
df = wd3
yy = find_outmost_convexhull_points(df, kappas = 4)
yy$outmost #the rows of 4 outermost points
df[yy$outmost,] #the 4 outermost points
yy$outmostall #all outermost cH rows
yy$outmostfrequency #their frequency
```

`find_outmost_partitioned_convexhull_points` *Function which finds the outermost convex hull points after making np samples and finding convex hull for each of them.*

Description

Function which finds the outermost convex hull points after making np samples and finding convex hull for each of them. To be used as initial solution in archetypal analysis

Usage

```
find_outmost_partitioned_convexhull_points(df, kappas, np = 10,
  nworkers = NULL)
```

Arguments

df	The data frame with dimensions n x d
kappas	The number of archetypes
np	The number of partitions that will be used (or the number of samples)
nworkers	The number of logical processors that will be used

Value

A list with members:

1. outmost, the first kappas most frequent outermost points as rows of data frame
2. outmostall, all the outermost points that have been found as rows of data frame
3. outmostfrequency, a matrix with frequency and cumulative frequency for outermost rows

See Also

find_furthestsum_points, find_outmost_projected_convexhull_points,

find_outmost_convexhull_points & find_outmost_points

Examples

```
data("wd2") #2D demo
df = wd2
yy = find_outmost_partitioned_convexhull_points(df, kappas = 3, nworkers = 1)
yy$outmost #the rows of 3 outermost points
df[yy$outmost,] #the 3 outermost points
yy$outmostall #all outermost rows
yy$outmostfrequency #their frequency
```

---

find_outmost_points *Function which finds the outermost points in order to be used as initial solution in archetypal analysis*

Description

Function which finds the outermost points in order to be used as initial solution in archetypal analysis

Usage

```
find_outmost_points(df, kappas)
```

Arguments

df	The data frame with dimensions n x d
kappas	The number of archetypes

Value

A list with members:

1. outmost, the first kappas most frequent outermost points as rows of data frame
2. outmostall, all the outermost points that have been found as rows of data frame
3. outmostfrequency, a matrix with frequency and cumulative frequency for outermost rows

Warning

This is a rather naive way to find the outermost points of a data frame and it should be used with caution since for a n x d matrix we need in general 8 n^2/(2^30) GB RAM for numeric case. Check your machine and use it. As a rule of thumb we advice its usage for n less or equal than 20000.

See Also

find_furthestsum_points, find_outmost_convexhull_points,

find_outmost_projected_convexhull_points,

and find_outmost_partitioned_convexhull_points

Examples

```
data("wd2") #2D demo
df = wd2
yy = find_outmost_points(df,kappas=3)
yy$outmost #the rows of 3 outmost points
yy$outmostall #all outmost found
yy$outmostfrequency #frequency table for all
df[yy$outmost,] #the 3 outmost points
#
###
#
data("wd3") #3D demo
df = wd3
yy = find_outmost_points(df,kappas=4)
yy$outmost #the rows of 4 outmost points
yy$outmostall #all outmost found
yy$outmostfrequency #frequency table for all
df[yy$outmost,] #the 4 outmost points
```

---

find_outmost_projected_convexhull_points

Function which finds the outermost projected convex hull points in order to be used as initial solution in archetypal analysis

---

Description

Function which finds the outermost projected convex hull points in order to be used as initial solution in archetypal analysis.

Usage

```
find_outmost_projected_convexhull_points(df, kappas, npr = 2, rseed = NULL,
                                         doparallel = FALSE, nworkers = NULL,
                                         uniquerows = FALSE)
```

Arguments

df	The n x d data frame that will be used for Archetypal Analysis
kappas	The number of archetypes

npr	The dimension of the projected subspaces. It can be npr = 1 (then there are d such subspaces), or npr > 1 (then we have C(d,npr) different subspaces)
rseed	An integer to be used for the random seed if it will be necessary
doparallel	If it is set to TRUE, then parallel processing will be performed. That is absolutely required if n is very large and d>6.
nworkers	The number of logical processors that will be used for computing the projected convex hulls, which they are always C(d,npr).
uniquerows	If it is set to TRUE, then unique rows will be used for computing distance matrix and less resources will be needed.

Details

If npr = 1, then Convex Hull is identical with the range (min,max) for the relevant variable, otherwise the function uses the chull when npr = 2 and the convhulln for npr > 2. See [1] and [2] respectively for more details.

First all available projections are being considered and their Convex Hull are being computed. Then either the unique (if uniquerows = TRUE) or all (if uniquerows = FALSE) associated data rows form a matrix and finally by using dist we find the kappas most frequent outermost rows.
A special care is needed if the rows we have found are less than kappas. In that case, if a random sampling is necessary, the output usedrandoms informs us for the number of random rows and the rseed can be used for reproducibility.

Value

A list with members:

1. outmost, the first kappas most frequent outermost points as rows of data frame
2. outmostall, all the outermost points that have been found as rows of data frame
3. outmostfrequency, a matrix with frequency and cumulative frequency for outermost rows
4. usedrandom, an integer of randomly chosen rows, if it was necessary to complete the number of kappas rows
5. chprojections, all the Convex Hulls of the different C(d,npr) projections, i.e. the coordinate projection subspaces
6. projected, a data frame with rows the unique points that have been projected in order to create the relevant Convex Hulls of coordinate projection subspaces

References

[1] Eddy, W. F. (1977). Algorithm 523: CONVEX, A new convex hull algorithm for planar sets. ACM Transactions on Mathematical Software, 3, 411-412. doi: 10.1145/355759.355768.

[2] Barber, C.B., Dobkin, D.P., and Huhdanpraa, H.T., "The Quickhull algorithm for convex hulls" ACM Trans. on Mathematical Software, 22(4):469-483, Dec 1996, http://www.qhull.org

See Also

find_furthestsum_points, find_outmost_convexhull_points

find_outmost_partitioned_convexhull_points & find_outmost_points

Examples

```
#
data("wd2") #2D demo
df = wd2
yy = find_outmost_projected_convexhull_points(df, kappas = 3)
yy$outmost #the rows of 3 outmost projected convexhull points
yy$outmostall #all outmost found
yy$outmostfrequency #frequency table for all
yy$usedrandom #No random row was used
yy$chprojections #The Convex Hull of projection (one only here)
yy$projected #the 9 unique points that created the one only CH
df[yy$outmost,] #the 3 outmost projected convexhull points
#
###
#
data("wd3") #3D demo
df = wd3
yy = find_outmost_projected_convexhull_points(df, kappas = 4)
yy$outmost #the rows of 4 outmost projected convexhull points
yy$outmostall #all outmost found
yy$outmostfrequency #frequency table for all
yy$usedrandom #No random row was used
yy$chprojections #All the Convex Hulls of projections top coordinate planes
yy$projected #the 14 unique points that created all CHs
df[yy$outmost,] #the 4 outmost projected convexhull points
#
```

`find_pcha_optimal_parameters`

Finds the optimal updating parameters to be used for the PCHA algorithm

---

Description

After creating a grid on the space of (mu_up, mu_down) it runs `archetypal` by using a given `method` & other running options passed by ellipsis (...) and finally finds those values which minimize the SSE at the end of `testing_iters` iterations (default=10).

Usage

```
find_pcha_optimal_parameters(df, kappas, method = "projected_convexhull",
testing_iters = 10, nworkers = NULL, nprojected = 2, npartition = 10,
nfurthest = 100, sortrows = FALSE,
mup1 = 1.1, mup2 = 2.50, mdown1 = 0.1, mdown2 = 0.5, nmup = 10, nmdown = 10,
rseed = NULL, plot = FALSE, ...)
```

Arguments

df	The data frame with dimensions n x d
kappas	The number of archetypes
method	The method that will be used for computing initial approximation: 1. projected_convexhull, see find_outmost_projected_convexhull_points 2. convexhull, see find_outmost_convexhull_points 3. partitioned_convexhull, see find_outmost_partitioned_convexhull_points 4. furthestsum, see find_furthestsum_points 5. outmost, see find_outmost_points 6. random, a random set of kappas points will be used
testing_iters	The maximum number of iterations to run for every pair (mu_up, mu_down) of parameters
nworkers	The number of logical processors that will be used for parallel computing (usually it is the double of available physical cores)
nprojected	The dimension of the projected subspace for find_outmost_projected_convexhull_points
npartition	The number of partitions for find_outmost_partitioned_convexhull_points
nfurthest	The number of times that FurthestSum algorithm will be applied
sortrows	If it is TRUE, then rows will be sorted in find_furthestsum_points
mup1	The minimum value of mu_up, default is 1.1
mup2	The maximum value of mu_up, default is 2.5
mdown1	The minimum value of mu_down, default is 0.1
mdown2	The maximum value of mu_down, default is 0.5
nmup	The number of points to be taken for [mup1,mup2], default is 10
nmdown	The number of points to be taken for [mdown1,mdown2]
rseed	The random seed that will be used for setting initial A matrix. Useful for reproducible results
plot	If it is TRUE, then a 3D plot for (mu_up, mu_down, SSE) is created
...	Other arguments to be passed to function archetypal

Value

A list with members:

1. mu_up_opt, the optimal found value for muAup and muBup
2. mu_down_opt, the optimal found value for muAdown and muBdown
3. min_sse, the minimum SSE which corresponds to (mu_up_opt,mu_down_opt)
4. seed_used, the used random seed, absolutely necessary for reproducing optimal results
5. method_used, the method that was used for creating the initial solution
6. sol_initial, the initial solution that was used for all grid computations
7. testing_iters, the maximum number of iterations done by every grid computation

See Also

find_closer_points

Examples

```
{
data("wd25")
out = find_pcha_optimal_parameters(df = wd25, kappas = 5, rseed = 2020)
# Time difference of 30.91101 secs
# mu_up_opt mu_down_opt     min_sse
# 2.188889    0.100000    4.490980
# Run now given the above optimal found parameters:
aa = archetypal(df = wd25, kappas = 5,
                initialrows = out$sol_initial, rseed = out$seed_used,
                muAup = out$mu_up_opt, muAdown = out$mu_down_opt,
                muBup = out$mu_up_opt, muBdown = out$mu_down_opt)
aa[c("SSE", "varexpl", "iterations", "time" )]
# $SSE
# [1] 3.629542
#
# $varexpl
# [1] 0.9998924
#
# $iterations
# [1] 146
#
# $time
# [1] 21.96
# Compare it with a simple solution (time may vary)
aa2 = archetypal(df = wd25, kappas = 5, rseed = 2020)
aa2[c("SSE", "varexpl", "iterations", "time" )]
# $SSE
# [1] 3.629503
#
# $varexpl
# [1] 0.9998924
#
# $iterations
# [1] 164
#
# $time
# [1] 23.55
## Of course the above was a "toy example", if your data has thousands or million rows,
## then the time reduction is much more conspicuous.
# Close plot device:
dev.off()

}
```

`FurthestSum` *Application of FurthestSum algorithm in order to find an initial solution for Archetypal Analysis*

Description

The FurthestSum algorithm as was written by Morup and Hansen in Matlab, see [1] and it is based on [2]. The algorithm has been converted in order to use commonly used data frames in R.

Usage

```
FurthestSum(Y, kappas, irows, exclude = NULL)
```

Arguments

`Y`	The data frame with dimensions $n \times d$
`kappas`	The number of archetypes
`irows`	The initially used rows of data frame for starting algorithm
`exclude`	The rows of data frame that we want to exclude from being checked

Value

The vector of rows that constitute the initial FurthestSum solution

References

[1] Source: http://www.mortenmorup.dk/index_files/Page327.htm , last accessed 2019-06-07

[2] D.S. Hochbaum, D.B. Shmoys, A best possible heuristic for the k-center problem, Math. Oper. Res. 10(2) (1985) 180-184. https://doi.org/10.1287/moor.10.2.180

See Also

`find_furthestsum_points`

Examples

```
data("wd3") #3D demo
df = wd3
FurthestSum(df, kappas = 4, irows = sample(1:dim(df)[1],1))
```

gallupGPS6 *Gallup Global Preferences Study processed data set of six variables*

Description

A 76132 x 6 data frame derived from Gallup Global Preferences Study, see [1] and [2] for details. It can be used as a big data set example.

Usage

```
data("gallupGPS6")
```

Format

A data frame with 76132 complete observations on the following 6 variables.

`patience` a numeric vector

`risktaking` a numeric vector

`posrecip` a numeric vector

`negrecip` a numeric vector

`altruism` a numeric vector

`trust` a numeric vector

Details

Data processing:

1. The non complete rows have been removed
2. The duplicated rows have also been removed

Source

Individual data set was downloaded from https://www.briq-institute.org/global-preferences/downloads, last accessed 2019-12-17.

References

[1] Falk, A., Becker, A., Dohmen, T., Enke, B., Huffman, D., & Sunde, U. (2018). Global evidence on economic preferences. Quarterly Journal of Economics, 133 (4), 1645-1692.

[2] Falk, A., Becker, A., Dohmen, T. J., Huffman, D., & Sunde, U. (2016). The preference survey module: A validated instrument for measuring risk, time, and social preferences. IZA Discussion Paper No. 9674.

Examples

```
data(gallupGPS6)
summary(gallupGPS6)
```

`grouped_resample` *Function for performing simple or Dirichlet resampling*

Description

The function may be used for standard bootstrapping or for subsampling, see [1]. This function allows samples to be drawn with or without replacement, by groups and with or without Dirichlet weights, see [2]. This provides a variety of options for researchers who wish to correct sample biases, estimate empirical confidence intervals, and/or subsample large data sets.

Usage

```
grouped_resample(in_data = NULL, grp_vector = NULL, grp_matrix = NULL,
                 replace = FALSE, option = "Simple", number_samples = 1,
                 nworkers = NULL, rseed = NULL)
```

Arguments

`in_data`	The initial data frame that must be re-sampled. It must contain: 1. an ID variable 2. the variables of interest 3. a grouping variable
`grp_vector`	The grouping variable of the data frame, defined under the name 'group' for example
`grp_matrix`	A matrix that contains 1. the variable 'Group_ID' with entries all the available values of grouping variable 2. the variable 'Resample_Size' with the sizes for each sample that will be created per grouping value
`replace`	A logical input: TRUE/FALSE if replacement should be used or not, respectively
`option`	A character input with next possible values 1. "Simple", if we want to perform a simple re-sampling 2. "Dirichlet", if we want to perform a Dirichlet weighted re-sampling
`number_samples`	The number of samples to be created. If it is greater than one, then parallel processing is used.
`nworkers`	The number of logical processors that will be used for parallel computing (usually it is the double of available physical cores)
`rseed`	The random seed that will be used for sampling. Useful for reproducible results

Value

It returns a list of mumber_samples data frames with exactly the same variables as the initial one, except that group variable has now only the given value from input data frame.

Author(s)

David Midgley

References

[1] D. N. Politis, J. P. Romano, M. Wolf, Subsampling (Springer-Verlag, New York, 1999).

[2] Baath R (2018). bayesboot: An Implementation of Rubin's (1981) Bayesian Bootstrap. R package version 0.2.2, URL https://CRAN.R-project.org/package=bayesboot

See Also

dirichlet_sample

Examples

```
## Load absolute temperature data set:
data("AbsoluteTemperature")
df <- AbsoluteTemperature
## Find portions for climate zones
pcs <- table(df$z)/dim(df)[1]
## Choose the approximate size of the new sample and compute resample sizes
N <- round(sqrt(nrow(AbsoluteTemperature)))
resamplesizes=as.integer(round(N*pcs))
sum(resamplesizes)
## Create the grouping matrix
groupmat <- data.frame("Group_ID"=1:4,"Resample_Size"=resamplesizes)
groupmat
## Simple resampling:
resample_simple <- grouped_resample(in_data = df, grp_vector = "z",
                              grp_matrix = groupmat, replace = FALSE, option = "Simple",
                                  number_samples = 1, nworkers = NULL, rseed = 20191220)
cat(dim(resample_simple[[1]]),"\n")
## Dirichlet resampling:
resample_dirichlet <- grouped_resample(in_data = df, grp_vector = "z",
                             grp_matrix = groupmat, replace = FALSE, option = "Dirichlet",
                                  number_samples = 1, nworkers = NULL, rseed = 20191220)
cat(dim(resample_dirichlet[[1]]),"\n")
##
# ## Work in parallel and create many samples
# ## Choose a random seed
# nseed <- 20191119
# ## Simple
# reslist1 <- grouped_resample(in_data = df, grp_vector = "z", grp_matrix = groupmat,
#                           replace = FALSE, option = "Simple",
#                           number_samples = 10, nworkers = NULL,
#                           rseed = nseed)
```

```
# sapply(reslist1, dim)
# ## Dirichlet
# reslist2 <- grouped_resample(in_data = df, grp_vector = "z", grp_matrix = groupmat,
#                          replace = FALSE, option = "Dirichlet",
#                          number_samples = 10, nworkers = NULL,
#                          rseed = nseed)
# sapply(reslist2, dim)
# ## Check for same rows between 1st sample of 'Simple' and 1st sample of 'Dirichlet' ...
# mapply(function(x,y){sum(rownames(x)%in%rownames(y))},reslist1,reslist2)
#
```

`study_AAconvergence` *Function which studies the convergence of Archetypal Analysis when using the PCHA algorithm*

Description

First it finds an AA solution under given arguments while storing all iteration history (`save_history = TRUE`). Then it computes the LOWESS [1] of SSE and its relevant UIK point [2]. Study is performed for iterations after that point. The list of B-matrices and archetypes that were found are stored. The archetypes are being aligned, while the B-matrices are used for computing the used rows-weights, leading rows-weights and maybe percentage of used rows on Convex Hull. The Aitken SSE extrapolation plus the relevant error are computed. The order and rate of convergence are estimated. Finally a multi-plot panel is being created if asked.

Usage

```
study_AAconvergence(df, kappas, method = "projected_convexhull",
                    rseed = NULL, chvertices = NULL, plot = TRUE, ...)
```

Arguments

`df`	The data frame with dimensions n x d
`kappas`	The number of archetypes
`method`	The method that will be used for computing initial approximation: 1. projected_convexhull, see `find_outmost_projected_convexhull_points` 2. convexhull, see `find_outmost_convexhull_points` 3. partitioned_convexhull, see `find_outmost_partitioned_convexhull_points` 4. furthestsum, see `find_furthestsum_points` 5. outmost, see `find_outmost_points` 6. random, a random set of kappas points will be used
`rseed`	The random seed that will be used for setting initial A matrix. Useful for reproducible results.
`chvertices`	The vector of rows which represents the vertices for Convex Hull (if available)
`plot`	If it is TRUE, then a panel of useful plots is created
`...`	Other arguments to be passed to function `archetypal`, except `save_history` which must always be TRUE

Details

If we take natural logarithms at the next approximate equation

$$\epsilon_{n+1} = c\epsilon_n^p$$

for $n = 1, 2, 3, \ldots$, then we'll find

$$\log(\epsilon_{n+1}) = \log(c) + p\log(\epsilon_n)$$

Thus a reasonable strategy for estimating order p and rate c is to perform a linear regression on above errors, after a selected iteration. That is the output of `order_estimation` and `rate_estimation`.

Value

A list with members:

1. SSE, a vector of all SSE from all AA iterations
2. SSE_lowess, a vector of LOWESS values for SSE
3. UIK_lowess, the UIK point [2] of SSE_lowess
4. aitken, a data frame of Aitken [3] extrapolation and error for SSE after UIK_lowess iteration
5. order_estimation, the last term in estimating order of convergence, page 56 of [4], by using SSE after UIK_lowess iteration
6. rate_estimation, the last term in estimating rate of convergence, page 56 of [4], by using SSE after UIK_lowess iteration
7. significance_estimations, a data frame with standard errors and statistical significance for estimations
8. used_on_convexhull, the % of used rows which lie on Convex Hull (if given), as a sequence for iterations after UIK_lowess one
9. aligned_archetypes, the archetypes after UIK_lowess iteration are being aligned by using `align_archetypes_from_list`. The history of archetypes creation.
10. solution_used, the AA output that has been used. Some times useful, especially for big data.

References

[1] Cleveland, W. S. (1979) Robust locally weighted regression and smoothing scatterplots. J. Amer. Statist. Assoc. 74, 829–836.

[2] Christopoulos, Demetris T., Introducing Unit Invariant Knee (UIK) As an Objective Choice for Elbow Point in Multivariate Data Analysis Techniques (March 1, 2016). Available at SSRN: http://dx.doi.org/10.2139/ssrn.3043076

[3] Aitken, A. "On Bernoulli's numerical solution of algebraic equations", Proceedings of the Royal Society of Edinburgh (1926) 46 pp. 289-305.

[4] Atkinson, K. E.,An Introduction to Numerical Analysis, Wiley & Sons,1989

See Also

check_Bmatrix

Examples

```
{
# Load data "wd2"
data(wd2)
ch = chull(wd2)
sa = study_AAconvergence(df = wd2, kappas = 3, rseed = 20191119,
                         verbose = FALSE, chvertices = ch)
names(sa)
# [1] "SSE"                      "SSE_lowess"               "UIK_lowess"
# [4] "aitken"                   "order_estimation"         "rate_estimation"
# [7] "significance_estimations" "used_on_convexhull"       "aligned_archetypes"
# [10] "solution_used"
# sse=sa$SSE
# ssel=sa$SSE_lowess
sa$UIK_lowess
# [1] 36
# sa$aitken
sa$order_estimation
# [1] 1.007674
sa$rate_estimation
# [1] 0.8277613
sa$significance_estimations
#        estimation   std.error   t.value      p.value
# log(c) -0.1890305 0.014658947 -12.89523 5.189172e-12
# p       1.0076743 0.001616482 623.37475 3.951042e-50
# sa$used_on_convexhull
# sa$aligned_archetypes
data.frame(sa$solution_used[c("SSE","varexpl","iterations","time")])
#        SSE   varexpl iterations time
# 1 1.717538 0.9993186         62 8.39
# Close plot device:
dev.off()

}
```

---

wd2 *2D data set for demonstration purposes*

Description

A data frame of 100 2D points

Usage

```
data("wd2")
```

Format

matrix 100 x 2

Examples

```
# Creation of data set "wd2" from 3 corner-points:
p1 = c(1,2);p2 = c(3,5);p3 = c(7,3)
dp = rbind(p1,p2,p3);dp
set.seed(9102)
pts = t(sapply(1:100, function(i,dp){
  cc = runif(3)
  cc = cc/sum(cc)
  colSums(dp*cc)
},dp))
df = data.frame(pts)
colnames(df) = c("x","y")
head(df)
# Check all equal:
data(wd2)
all.equal(wd2,df)
# [1] TRUE
```

---

wd25 *2D data set created by 5 points for demonstration purposes*

Description

A data frame of 600 2D points

Usage

```
data("wd25")
```

Format

matrix 600 x 2

Examples

```
# Creation of data set "wd25" from 5 corner points:
set.seed(20191119)
p1 = c(3,2);p2 = c(4,6);p3 = c(7,8)
p4 = c(9,4);p5 = c(6,1)
dp = rbind(p1,p2,p3,p4,p5)
colnames(dp) = c('x','y')
pts=lapply(1:150, function(i,dp){
  c0 = runif(dim(dp)[1]);c0 = c0/sum(c0);pt0 = colSums(dp*c0)
  c1 = runif(3);c1 = c1/sum(c1);pt1 = colSums(dp[1:3,]*c1)
  c2 = runif(3);c2 = c2/sum(c2);pt2 = colSums(dp[c(4,5,1),]*c2)
```

```
  c3 = runif(3);c3 = c3/sum(c3);pt3 = colSums(dp[2:4,]*c3)
  rbind(pt0,pt1,pt2,pt3)
},dp)
df = do.call(rbind,pts)
rownames(df) = 1:dim(df)[1]
head(df)
# Check all equal
data("wd25")
all.equal(df,wd25)
# [1] TRUE
```

---

wd3 *3D data set for demonstration purposes*

Description

A data frame of 100 3D points

Usage

```
data("wd3")
```

Format

matrix 100 x 3

Examples

```
# Creation of data set "wd3" from 4 corner points:
p1 = c(3,0,0);p2 = c(0,5,0)
p3 = c(3,5,7);p4 = c(0,0,0)
# The data frame of generators
dp = data.frame(rbind(p1,p2,p3,p4))
colnames(dp) = c("x","y","z")
dp = dp[chull(dp),]
set.seed(9102)
df = data.frame(t(sapply(1:100, function(i,dp){
  cc = runif(4)
  cc = cc/sum(cc)
  colSums(dp*cc)
},dp)))
colnames(df) = c("x","y","z")
head(df)
# Check all.equal to "wd3"
data(wd3)
all.equal(df,wd3)
# [1] TRUE
```

Index