Package 'afCEC'

September 21, 2017

Type Package

Title Active function Cross-Entropy Clustering

Version 1.0.0

Date 2017-09-21

Author Krzysztof Byrski [aut, cre], Przemysław Spurek [ctb]

Maintainer Krzysztof Byrski <krzysiek.byrski@uj.edu.pl>

Description

Active Function Cross-Entropy Clustering (afCEC) partitions the n-dimensional data into the clusters by finding the parameters of the mixed generalized multivariate normal distribution, that optimally approximates the scattering of the data in the n-dimensional space, whose density function is of the form: p_1*N(mi_1,^sigma_1,sigma_1,f_1)+...+p_k*N(mi_k,^sigma_k,sigma_k,f_k). The above-mentioned generalization is performed by introducing so called ``f-adapted gaussian densities'' (i.e. the ordinary gaussian densities adapted by the ``active function''). Additionally, the Active Function Cross-Entropy Clustering performs the automatic reduction of the unnecessary clusters.

URL https://github.com/GeigenPrinzipal/afCEC

Encoding UTF-8

NeedsCompilation yes

License GPL (>= 2)

Imports Rcpp (>= 0.11.3), RcppArmadillo, rgl

LinkingTo Rcpp, RcppArmadillo, rgl

Depends Rcpp, RcppArmadillo, rgl

R topics documented:

`afCEC-package` *Active Function Cross-Entropy Clustering*

Description

Active Function Cross-Entropy Clustering (afCEC) partitions the n-dimensional data into the clusters by finding the parameters of the mixed generalized multivariate normal distribution, that optimally approximates the scattering of the data in the n-dimensional space, whose density function is of the form:

p_1*N(mi_1,^sigma_1,sigma_1,f_1)+...+p_k*N(mi_k,^sigma_k,sigma_k,f_k)

The above-mentioned generalization is performed by introducing so called "f-adapted gaussian densities" (i.e. the ordinary gaussian densities adapted by the "active function"). Additionally, the Active Function Cross-Entropy Clustering performs the automatic reduction of the unnecessary clusters. It's implemented in the form of a customizable function `afCEC`.

Details

Package:	afCEC
Type:	Package
Version:	1.0.0
Date:	2017-21-09
License:	GPL (>= 2)

Author(s)

Krzysztof Byrski

See Also

`afCEC`.

afCEC *afCEC*

Description

Performs the Active Function Cross-Entropy Clustering on the data set.

Usage

```
afCEC (
    points,
    maxClusters,
    initialLabels="k-means++",
    cardMin=0.01,
    costThreshold=-0.000001,
    minIterations=10,
    maxIterations=100,
    numberOfStarts=10,
    method="Hartigan",
    values="quadratic",
    interactive=FALSE
)
```

Arguments

points	A (#points x n) matrix of data containing n-dimensional points stored row-by-row.
maxClusters	Indicates the maximal number of clusters, that afCEC can partition the data into.
initialLabels	Initial labelling determining the data membership to the particular clusters. There are 3 options allowed: • "random" - causes the labelling to be randomly generated. • "k-means++" - causes the labelling to be generated using the k-means++ heuristics. • A (#points x numberOfStarts) matrix of type integer containing values in the range: 0...maxClusters - 1. The value x in the row i and column j indicates that in the j-th start, the i-th point will be initially assigned to the (x + 1)-th cluster. The defalt value is "k-means++".
cardMin	Value so that if the number of points in particular cluster relatively to the size of the whole data drops bellow, then that cluster gets removed and it's data is assigned to the other clusters. The default value is 0.01.
costTreshold	Negative value so that if the difference between the calculated cost in the subsequent iterations is greater than it then the afCEC terminates returning the solution from the terminated iteration as the final solution (in the present start), provided that at least minIterations already passed. The default value is -0.000001;
minIterations	Value indicating the minimal number of iterations needed before afCEC can terminate, provided that no error occurred. The default value is 10.
maxIterations	Value indicating the maximal number of iterations in one start, that afCEC cannot exceed. The default value is 100.

`numberOfStarts` Value indicating the number of runs of the algorithm to be performed. The best solution is chosen out of the solutions given in the particular steps, so that it has the lowest value of the cost function among them. The default value is 10.

`method` Heuristics used to perform the clustering. There are 2 options allowed:

- "Lloyd" - indicates that the Lloyd heuristics will be used to perform the clustering.
- "Hartigan" - indicates that the Hartigan heuristics will be used to perform the clustering.

The default value is "Hartigan".

`values` Definition of the active function family used to perform the clustering. There are 3 options allowed:

- "quadratic" - indicates that the function family of the form:

  f(x_1,...,x_(n-1))=a_1*x_1^2+...+a_(n-1)*x_(n-1)^2+a_n*x_1+...+a_(2*n-2)*x_(n-1)+a_(2*n-1)*1,

  where n is the dimensionality of the data and a_i are the optimal coefficients determined by afCEC using the linear least square fitting, will be used to perform the clustering.
- "String containing the user-defined formula. See more in `User-defined formulas`.
- A ((m*n) x #points) matrix containing the values of the particular components of the active function on the data set, placed according to the following layout (row by row):

  First m rows:

  [f_1(x_1_2,...,x_1_n), f_1(x_2_2,...,x_2_n), ..., f_1(x_#points_2,...,x_#points_n)]
  [f_2(x_1_2,...,x_1_n), f_2(x_2_2,...,x_2_n), ..., f_2(x_#points_2,...,x_#points_n)]
  ...
  [f_m(x_1_2,...,x_1_n), f_m(x_2_2,...,x_2_n), ..., f_m(x_#points_2,...,x_#points_n)]

  Second m rows:

  [f_1(x_1_1,x_1_3,...,x_1_n), f_1(x_2_1,x_2_3,...,x_2_n), ...]
  [f_2(x_1_1,x_1_3,...,x_1_n), f_2(x_2_1,x_2_3,...,x_2_n), ...]
  ...
  [f_m(x_1_1,x_1_3,...,x_1_n), f_m(x_2_1,x_2_3,...,x_2_n), ...]

  Last m rows:

  [f_1(x_1_1,...,x_1_(n-1)), f_1(x_2_1,...,x_2_(n-1)), ...]
  [f_2(x_1_1,...,x_1_(n-1)), f_2(x_2_1,...,x_2_(n-1)), ...]
  ...
  [f_m(x_1_1,...,x_1_(n-1)), f_m(x_2_1,...,x_2_(n-1)), ...],

  where: n - dimensionality of the data, x_i_j - j-th coordinate of the i-th point of data, m - number of components of the active function.

  In the foregoing case, the active function family consists of the functions of the form:

f(x_1,...,x_(n-1))=a_1*f_1(x_1,...,x_(n-1))+...+a_m*f_m(x_1,...,x_(n-1)),

where n is the dimensionality of the data and a_i are the optimal coefficients determined by afCEC using the linear least square fitting.

The default value is "quadratic".

Remarks:

In the case of the third way of defining the function, the returned object doesn't contain the information about the means coordinates corresponding to the active direction. Moreover, the plotting capabilities of the afCEC package are severely impaired in that case. See `plot`.

`interactive` Indicates if the algorithm runs in the "interactive" mode. The "interactive" mode allows to track the intermediate steps of the afCEC. Instead of one object of the afCEC class, the whole list of such objects is returned, where each item of the list corresponds to the intermediate step of the algorithm. See `value` section for more details. The default value is FALSE.

Value

- Empty list - if clustering failed.
- Object of class afCEC containing the parameters of the best fitted mixed generalized multivariate normal distribution - if argument interactive=FALSE and clustering succeded.
- List of k objects of class afCEC containing the parameters of the best fitted mixed generalized multivariate normal distribution across the k subsequent steps of the algorithm - if argument interactive=TRUE and clustering succeeded.

See Also

`plot`

Examples

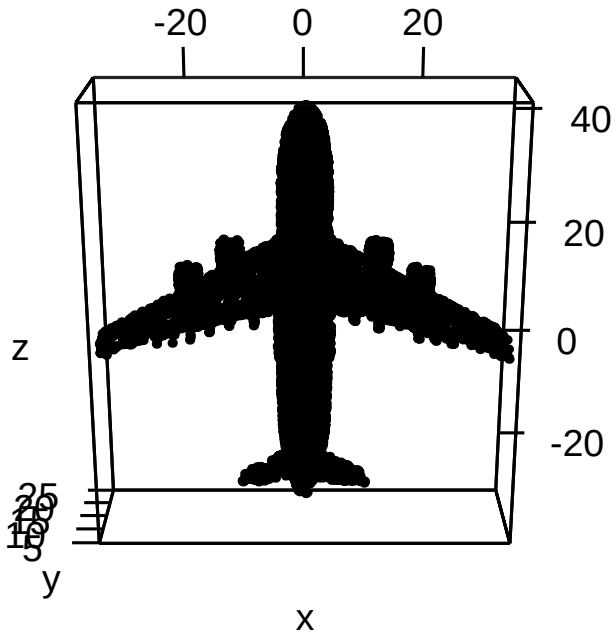
```
# The following three examples demonstrate three equivalent ways of passing the same
# 3D quadratic active function to the afCEC routine:

# 1) Using "quadratic" value (default):

library(afCEC);
data(airplane);
result <- afCEC(airplane, 17);

# what is equivalent to:

library(afCEC);
data(airplane);
result <- afCEC(airplane, 17, values="quadratic");

# 2) Using the string with the user-defined formula:

library(afCEC);
data(airplane);
formula = paste(
    "f:R^2 -> R^5\n",
```

```
    "f^1(x) = x(1)^2\n",
    "f^2(x) = x(2)^2\n",
    "f^3(x) = x(1)\n",
    "f^4(x) = x(2)\n",
    "f^5(x) = 1\n",
    sep=""
);
result <- afCEC(airplane, 17, values=formula);

# 3) Using the matrix containing the explicit values of the active function across the
#    subsequent dimensions:

library(afCEC);
data(airplane);
values = matrix(rep(0, 5*3*dim(airplane)[1]), 5*3, dim(airplane)[1]);
for (i in 1:dim(airplane)[1]) {
    tmp <- airplane[i,2:3];
    for (j in 1:3) {
        values[((j - 1) * 5) + 1, i] <- tmp[1]^2;
        values[((j - 1) * 5) + 2, i] <- tmp[2]^2;
        values[((j - 1) * 5) + 3, i] <- tmp[1];
        values[((j - 1) * 5) + 4, i] <- tmp[2];
        values[((j - 1) * 5) + 5, i] <- 1;
        if (j < 3) tmp[j] <- airplane[i,j];
    }
}
result <- afCEC(airplane, 17, values=values);
```

`airplane` *airplane*

Description

A 10000 x 3 matrix of three dimensional points uniformly distributed on the surface of the airplane model.



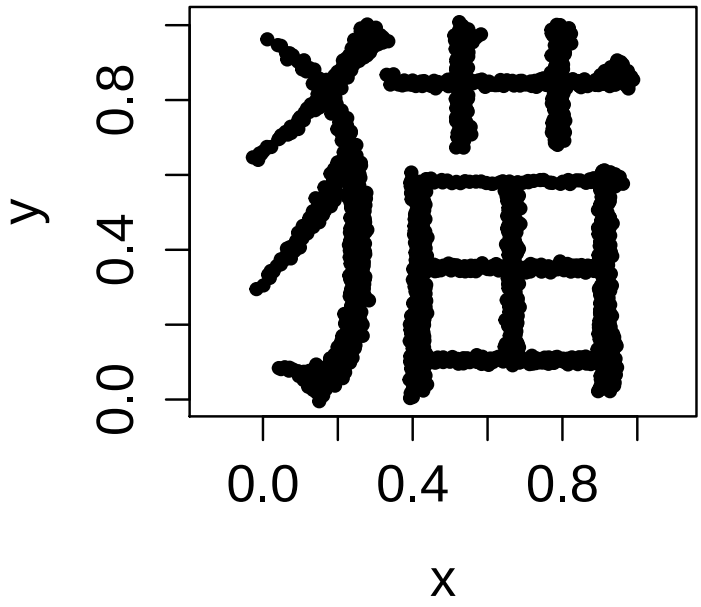
Examples

```
library(afCEC);
data(airplane);
plot3d(airplane);
```

`cat` *cat*

Description

A 1385 x 2 matrix of two dimensional points forming the chinese "cat" letter.



Examples

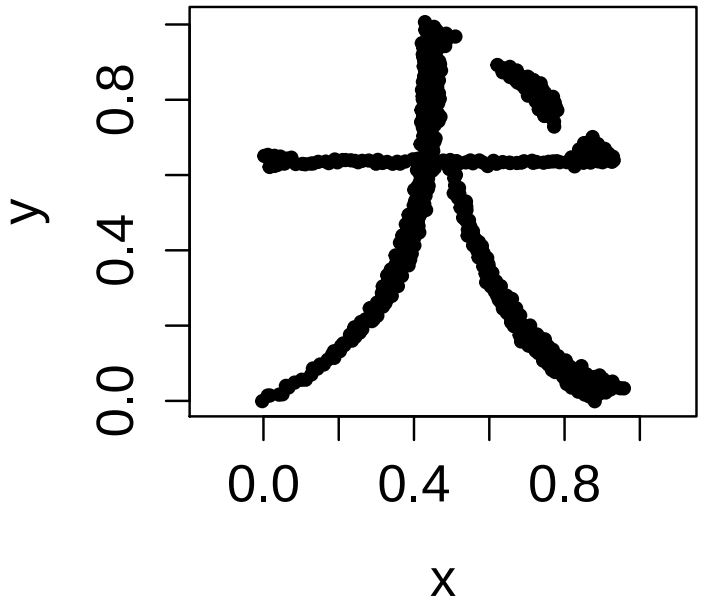
```
library(afCEC);
data(cat);
plot(cat);
```

---

dog *dog*

Description

A 623 x 2 matrix of two dimensional points forming the chinese "dog" letter.



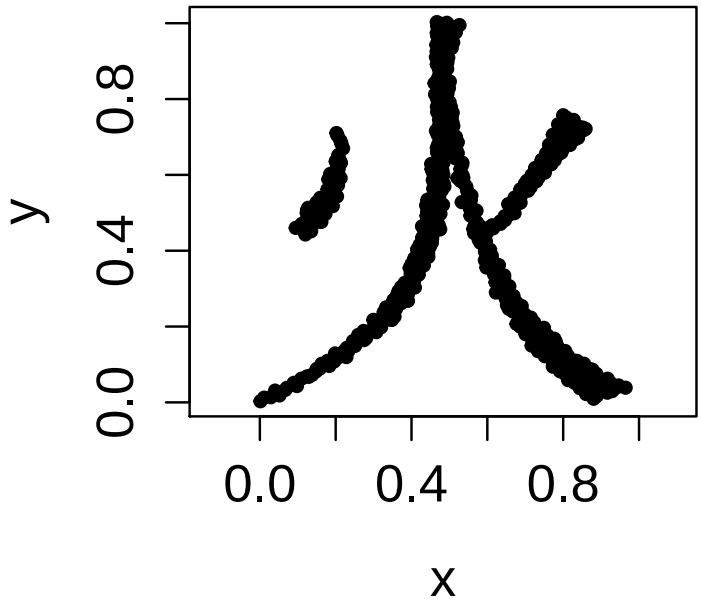
Examples

```
library(afCEC);
data(dog);
plot(dog);
```

`fire` *fire*

Description

A 609 x 2 matrix of two dimensional points forming the chinese "fire" letter.



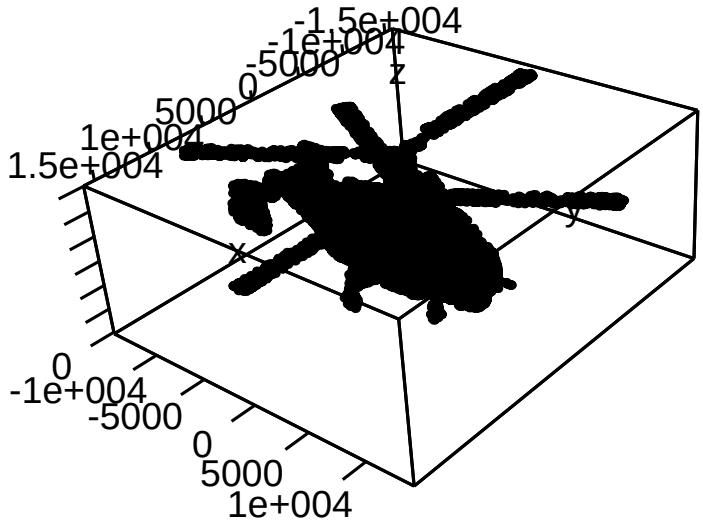
Examples

```
library(afCEC);
data(fire);
plot(fire);
```

`helicopter` *helicopter*

Description

A 10000 x 3 matrix of three dimensional points uniformly distributed on the surface of the helicopter model.



Examples

```
library(afCEC);
data(helicopter);
plot3d(helicopter);
```

`JaccardIndex` *JaccardIndex*

Description

Computes the Jaccard similarity index between two clusterings determined by the indices of the clusters the particular data points belong to, given by the parameters: `labels1` and `labels2` respectively. Both parameters are the equal-length vectors of the type integer with values greater than or equal to 0. If one of the above-mentioned conditions isn't met, the exception is thrown and the function returns with errors.

Usage

```
JaccardIndex(labels1, labels2)
```

Value

A value in the range [0,1]

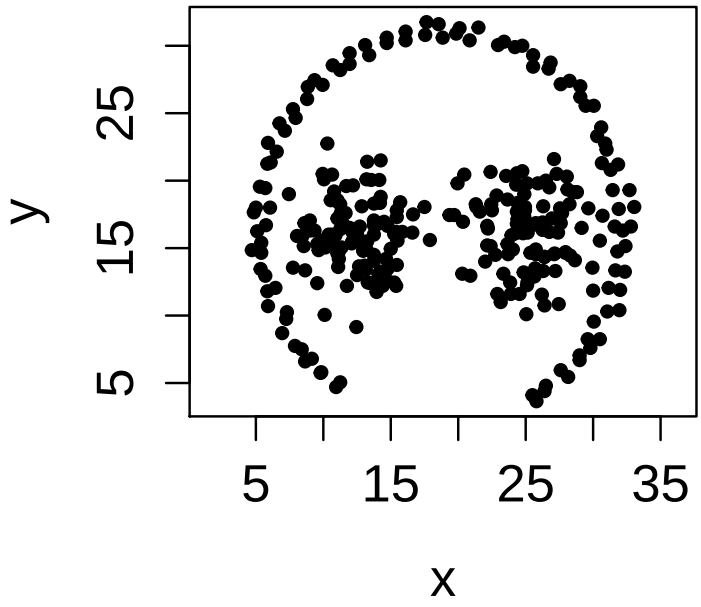
Examples

```
library(afCEC);
points1 <- SampleBallUniform(1000,2);
points2 <- (SampleBallUniform(2000,2)*2) + matrix(rep(c(2,0),2000),2000,2,byrow=TRUE);
points <- rbind(points1,points2);
labels <- c(rep(0,1000),rep(1,2000));
result <- afCEC(points,2);
ri <- JaccardIndex(labels,result$labels);
print(
    paste(
       "The Jaccard index between the model clustering and the clustering obtained by the ",
        "afCEC function is: ",
        ri,
        ".",
        sep=""
    )
);
```

`pathbased` *pathbased*

Description

A 300 x 2 matrix of two dimensional points forming the incomplete circle and two filled circles inside.



Examples

```
library(afCEC);
data(pathbased);
plot(pathbased);
```

`plot` *plot*

Description

Plots the chart showing the clustering of the data. Depending on the dimensionality of the data passed to the afCEC function, it draws either 2D or 3D chart. The function doesn't work with the other dimensions.

Remarks:

- The particular items are drawn cluster-by-cluster in the following order:
  - points
  - means
  - curves / surfaces
  - ellipses / ellipsoids of confidence

- The functionality concerning drawing the means (see the `draw_means` argument description for more details), ellipses / ellipsoids of confidence (see the `draw_ellipsoids` argument description for more details) and curves / surfaces (see the `draw_surfaces` argument description for more details) doesn't work if object x was retrieved after clustering the data with the active function family defined by the matrix of the active function components values on the data set. In that case, the above-mentioned arguments as well as the other ones related to them (for example `meansSize2D`) are ignored and only data points are drawn on the chart. See afCEC.

Usage

```
plot (
    x,
    draw_points=TRUE,
    draw_means=TRUE,
    draw_ellipsoids=TRUE,
    draw_surfaces=TRUE,
    confidence=0.95,
    grid_resolution=32,

    pointsSize2D=1, pointsColor2D="cluster",
    meansSize2D=2, meansColor2D="black",
    ellipsesHeight2D=1, ellipsesColor2D="black",
    surfacesHeight2D=1, surfacesColor2D="black",
    XLabel2D="X", YLabel2D="Y",

    pointsAlpha3D=1, pointsSize3D=0.01, pointsColor3D="cluster",
    meansAlpha3D=0.5, meansSize3D=0.01, meansColor3D="black",
    ellipsoidsAlpha3D=0.25, ellipsoidsColor3D="cluster",
    surfacesAlpha3D=0.5, surfacesColor3D="cluster",
    XLabel3D="X", YLabel3D="Y", ZLabel3D="Z"
)
```

Arguments

`x`	Object of class afCEC returned from the afCEC function.
`draw_points`	If the value is TRUE, the function draws the points belonging to the particular clusters. The default value is TRUE.
`draw_means`	If the value is TRUE, the function draws the means of the clusters as the big black dots / spheres. The default value is TRUE.
`draw_ellipsoids`	If the value is TRUE, the function draws the curved ellipses / ellipsoids of confidence of the particular clusters. The default value is TRUE.
`draw_surfaces`	If the value is TRUE, the function draws the curves / surfaces corresponding to the subspace spanned by the inactive axes of the particular clusters with respect to the curvilinear coordinate system determined by the active function with 0's in the means of the clusters. The default value is TRUE.
`confidence`	Determines the percentile of data belonging to the particular clusters, the corresponding ellipses / ellipsoids of confidence should contain. For example, if the value 0.5 is specified, then, for each cluster, it's ellipse / ellipsoid of confidence will contain 0.5 * 100 [%] of assigned points. The default value is 0.95.

`grid_resolution`
: Determines the grid resolution using to approximate the curves / surfaces drawn in the plot function. The default value is 32.

`pointsSize2D`
: Determines the size of the points (in pixels) drawn in the plot function. The argument is used only in the case of the 2D data. The default value is 1.

`pointsColor2D`
: Determines the color of the points drawn in the plot function. It can hold the following values:

  - Any color format acceptable by the standard R plot function. In that case the color is fixed across the particular clusters.
  - The "cluster" string indicating, that the plot function will use unique colors for points belonging to the different clusters.

  The argument is used only in the case of the 2D data. The default value is "cluster".

`meansSize2D`
: Determines the size of the means (in pixels) drawn in the plot function. The argument is used only in the case of the 2D data. The default value is 2.

`meansColor2D`
: Determines the color of the means drawn in the plot function. It can hold the following values:

  - Any color format acceptable by the standard R plot function. In that case the color is fixed across the particular clusters.
  - The "cluster" string indicating, that the plot function will use unique colors for means of the different clusters.

  The argument is used only in the case of the 2D data. The default value is "black".

`ellipsesHeight2D`
: Determines the thickness of the ellipses (in pixels) drawn in the plot function. The argument is used only in the case of the 2D data. The default value is 1.

`ellipsesColor2D`
: Determines the color of the ellipses drawn in the plot function. It can hold the following values:

  - Any color format acceptable by the standard R plot function. In that case the color is fixed across the particular clusters.
  - The "cluster" string indicating, that the plot function will use unique colors for ellipses of confidence of the different clusters.

  The argument is used only in the case of the 2D data. The default value is "black".

`surfacesHeight2D`
: Determines the thickness of the curves (in pixels) drawn in the plot function (see the `draw_surfaces` argument description for more details). The argument is used only in the case of the 2D data. The default value is 1.

`surfacesColor2D`
: Determines the color of the curves drawn in the plot function. It can hold the following values:

  - Any color format acceptable by the standard R plot function. In that case the color is fixed across the particular clusters.
  - The "cluster" string indicating, that the plot function will use unique colors for curves (see the `draw_surfaces` argument description for more details) belonging the different clusters.

The argument is used only in the case of the 2D data. The default value is "black".

`XLabel2D` Determines the label on the X axis of the chart drawn by the plot function. The argument is used only in the case of the 2D data. The default value is "X".

`YLabel2D` Determines the label on the Y axis of the chart drawn by the plot function. The argument is used only in the case of the 2D data. The default value is "Y".

`pointsAlpha3D` Determines the opacity of the points drawn in the plot function. The argument is used only in the case of the 3D data. The default value is 1.

`pointsSize3D` Determines the size of the points drawn in the plot function. The size is expressed as the fraction of the maximal extent of the bounding box containing the whole of the data points. The argument is used only in the case of the 3D data. The default value is 0.01.

`pointsColor3D` Determines the color of the points drawn in the plot function. It can hold the following values:

- Any color format acceptable by the standard R plot function. In that case the color is fixed across the particular clusters.
- The "cluster" string indicating, that the plot function will use unique colors for points belonging to the different clusters.

The argument is used only in the case of the 3D data. The default value is "cluster".

`meansAlpha3D` Determines the opacity of the means drawn in the plot function. The argument is used only in the case of the 3D data. The default value is 0.5.

`meansSize3D` Determines the size of the means drawn in the plot function. The size is expressed as the fraction of the maximal extent of the bounding box containing the whole of the data points. The argument is used only in the case of the 3D data. The default value is 0.01.

`meansColor3D` Determines the color of the means drawn in the plot function. It can hold the following values:

- Any color format acceptable by the standard R plot function. In that case the color is fixed across the particular clusters.
- The "cluster" string indicating, that the plot function will use unique colors for points belonging to the different clusters.

The argument is used only in the case of the 3D data. The default value is "black".

`ellipsoidsAlpha3D`

Determines the opacity of the ellipsoids drawn in the plot function. The argument is used only in the case of the 3D data. The default value is 0.25.

`ellipsoidsColor3D`

Determines the color of the ellipsoids drawn in the plot function. It can hold the following values:

- Any color format acceptable by the standard R plot function. In that case the color is fixed across the particular clusters.
- The "cluster" string indicating, that the plot function will use unique colors for ellipses of confidence of the different clusters.

The argument is used only in the case of the 3D data. The default value is "cluster".

surfacesAlpha3D
: Determines the opacity of the surfaces drawn in the plot function. The argument is used only in the case of the 3D data. The default value is 0.5.

surfacesColor3D
: Determines the color of the surfaces drawn in the plot function. It can hold the following values:
  - Any color format acceptable by the standard R plot function. In that case the color is fixed across the particular clusters.
  - The "cluster" string indicating, that the plot function will use unique colors for ellipses of confidence of the different clusters.

  The argument is used only in the case of the 3D data. The default value is "cluster".

XLabel3D
: Determines the label on the X axis of the chart drawn by the plot function. The argument is used only in the case of the 3D data. The default value is "X".

YLabel3D
: Determines the label on the Y axis of the chart drawn by the plot function. The argument is used only in the case of the 3D data. The default value is "Y".

ZLabel3D
: Determines the label on the Y axis of the chart drawn by the plot function. The argument is used only in the case of the 3D data. The default value is "Z".

Value

None.

See Also

afCEC

Examples

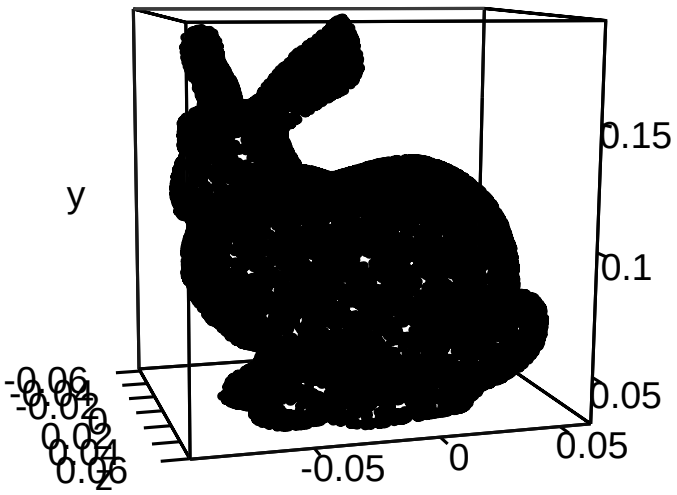
```
library(afCEC);
data(airplane);
result <- afCEC(airplane, 17);
plot(result);
```

---

rabbit *rabbit*

Description

A 10000 x 3 matrix of three dimensional points uniformly distributed on the surface of the rabbit model.



Examples

```
library(afCEC);
data(rabbit);
plot3d(rabbit);
```

RandIndex *RandIndex*

Description

Computes the Rand similarity index between two clusterings determined by the indices of the clusters the particular data points belong to, given by the parameters: `labels1` and `labels2` respectively. Both parameters are the equal-length vectors of the type integer with values greater than or equal to 0. If one of the above-mentioned conditions isn't met, the exception is thrown and the function returns with errors.

Usage

```
RandIndex(labels1, labels2)
```

Value

A value in the range [0,1]

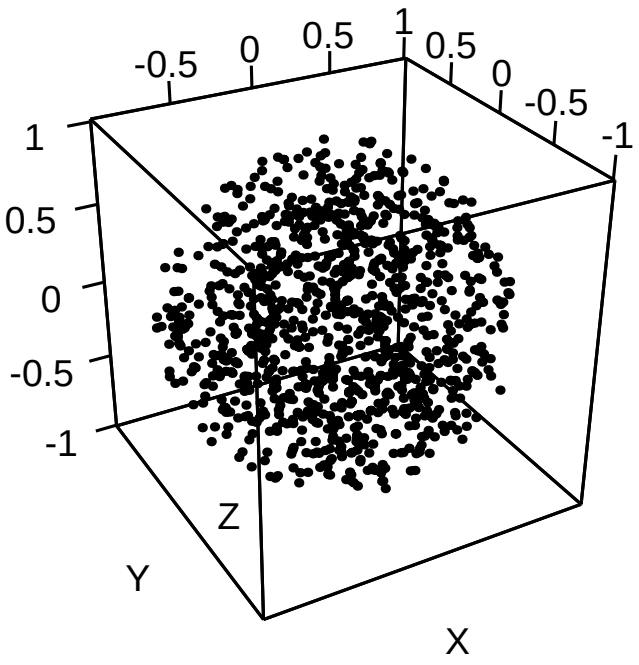
Examples

```
library(afCEC);
points1 <- SampleBallUniform(1000,2);
points2 <- (SampleBallUniform(2000,2)*2) + matrix(rep(c(2,0),2000),2000,2,byrow=TRUE);
points <- rbind(points1,points2);
labels <- c(rep(0,1000),rep(1,2000));
result <- afCEC(points,2);
ri <- RandIndex(labels,result$labels);
print(
    paste(
        "The Rand index between the model clustering and the clustering obtained by the ",
        "afCEC function is: ",
        ri,
        ".",
        sep=""
    )
);
```

SampleBallUniform *SampleBallUniform*

Description

Generates n uniformly distributed points inside a dim-dimensional unit ball. The chart below depicts the result of executing the code placed in the "examples" section:



Usage

```
SampleBallUniform(n, dim)
```

Value

A (n x dim) matrix of dim-dimensional points stored row-by-row

Examples

```
library(afCEC);
points <- SampleBallUniform(1000, 3);
plot3d(points, xlab="X", ylab="Y", zlab="Z");
```

SampleMeshUniform *SampleMeshUniform*

Description

Generates n uniformly distributed points on the surface of the mesh of the 3D object in the *.obj format with the path given by the parameter path. The chart below depicts the result of executing the code placed in the "examples" section:

Usage

```
SampleMeshUniform(path, n)
```

Value

A (n x 3) matrix of three-dimensional points stored row-by-row

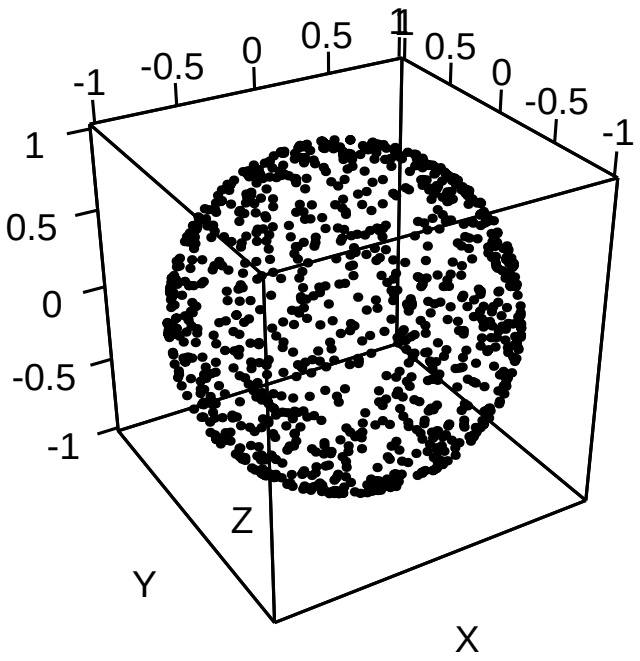
Examples

```
library(afCEC);
points <- SampleMeshUniform("neptune.obj", 10000);
plot3d(points, aspect=FALSE, axes=FALSE, xlab="", ylab="", zlab="");
axes3d(box=TRUE, xlen=0, ylen=0, zlen=0);
```

SampleSphereUniform *SampleSphereUniform*

Description

Generates n uniformly distributed points on a dim-dimensional unit sphere embeded in the (dim+1)-dimensional space. The chart below depicts the result of executing the code placed in the "examples" section:



Usage

```
SampleSphereUniform(n, dim)
```

Value

A (n x dim+1) matrix of (dim+1)-dimensional points stored row-by-row

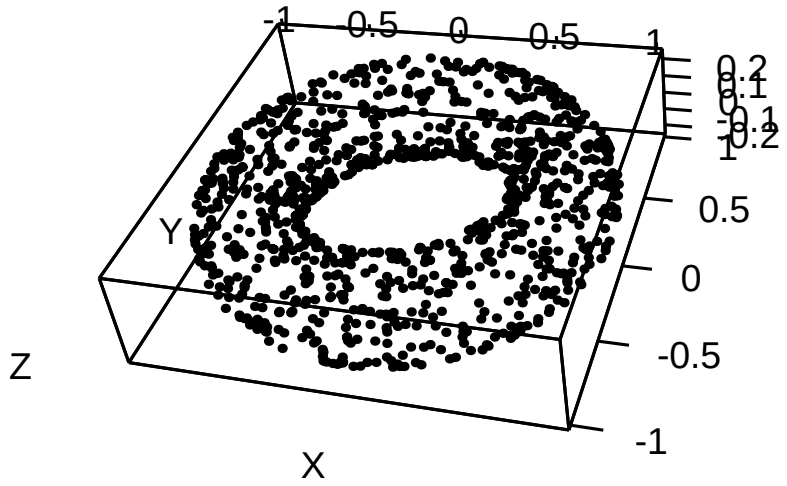
Examples

```
library(afCEC);
points <- SampleSphereUniform(1000, 2);
plot3d(points, xlab="X", ylab="Y", zlab="Z");
```

SampleTorusUniform *SampleTorusUniform*

Description

Generates n approximately uniformly distributed points on the surface of the three-dimensional torus obtained by revolution of the circle with the radius r around the Z axis. The parameter R corresponds to the distance from the torus center to the center of any circular cross section. The chart below depicts the result of executing the code placed in the "examples" section:



Usage

```
SampleTorusUniform(R, r, n)
```

Value

A (n x 3) matrix of three-dimensional points stored row-by-row

Remarks

The function uses the inverse cumulative distribution function method to obtain the approximately uniformly distributed samples. As one of the marginal cdfs corresponding to one of the parameters of the torus parametric equation doesn't have the inverse function that can be expressed in terms of the elementary functions, the routine `SampleTorusUniform` uses its reasonable approximation. Therefore, the resulting sampling is exact only for approximately close values of `R` and `r`.

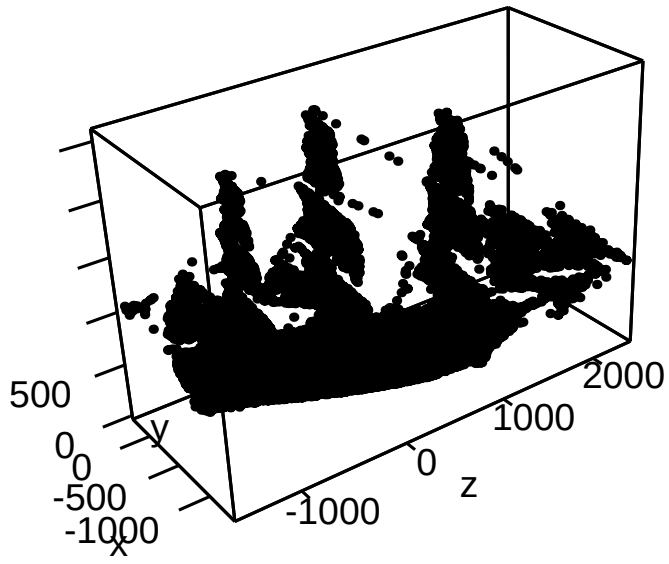
Examples

```
library(afCEC);
points <- SampleTorusUniform(0.75, 0.25, 1000);
plot3d(points, aspect=FALSE, xlab="X", ylab="Y", zlab="Z");
```

`ship` *ship*

Description

A 10000 x 3 matrix of three dimensional points uniformly distributed on the surface of the ship model.



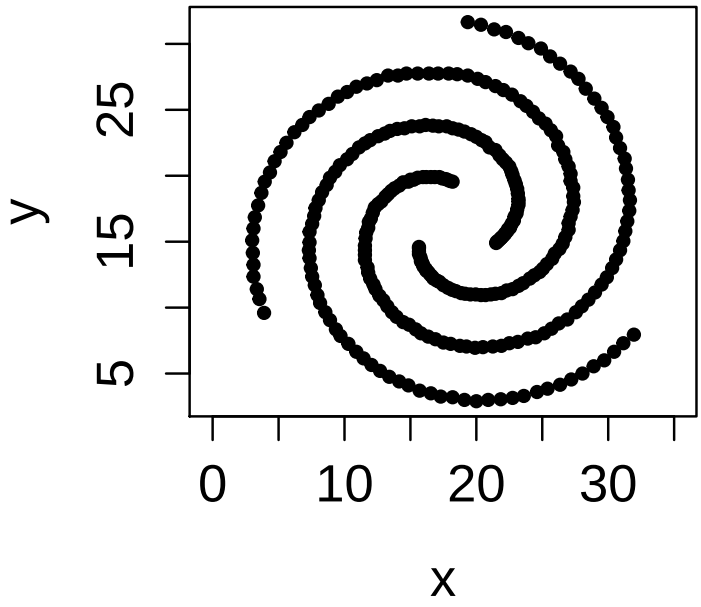
Examples

```
library(afCEC);
data(ship);
plot3d(ship);
```

`spiral` *spiral*

Description

A 312 x 2 matrix of two dimensional points forming the spiral.



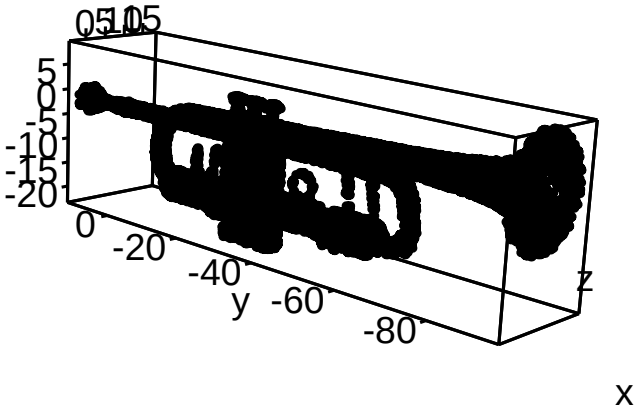
Examples

```
library(afCEC);
data(spiral);
plot(spiral);
```

`trumpet` *trumpet*

Description

A 10000 x 3 matrix of three dimensional points uniformly distributed on the surface of the trumpet model.



Examples

```
library(afCEC);
data(trumpet);
plot3d(trumpet);
```

Index