Package ‘clustlearn’

September 14, 2023

Title Learn Clustering Techniques Through Examples and Code

Version 1.0.0

Description Clustering methods, which (if asked) can provide step-by-step explanations of the algorithms used, as described in Ezugwu et. al., (2022) <doi:10.1016/j.engappai.2022.104743>; and datasets to test them on, which highlight the strengths and weaknesses of each technique, as presented in the clustering section of 'scikit-learn' (Pedregosa et al., 2011) <https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html>.

URL https://github.com/Ediu3095/clustlearn

BugReports https://github.com/Ediu3095/clustlearn/issues

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.3

Depends R (>= 4.3.0)

Imports proxy (>= 0.4-27), cli (>= 3.6.1)

Suggests deldir (>= 1.0-9)

LazyData true

NeedsCompilation no

Author Eduardo Ruiz Sabajanes [aut, cre],
Juan Jose Cuadrado Gallego [ctb]
(<https://orcid.org/0000-0001-8178-5556>),
Universidad de Alcala [cph]

Maintainer Eduardo Ruiz Sabajanes <eduardo.ruizs@edu.uah.es>

Repository CRAN

Date/Publication 2023-09-14 19:00:02 UTC

R topics documented:

agglomerative_clustering

Agglomerative Hierarchical Clustering

Description

Perform a hierarchical agglomerative cluster analysis on a set of observations

Usage

```
agglomerative_clustering(
  data,
  proximity = "single",
  details = FALSE,
  waiting = TRUE,
  ...
)
```

Arguments

`data` a set of observations, presented as a matrix-like object where every row is a new observation.

`proximity` the proximity definition to be used. This should be one of `"single"` (minimum/single linkage), `"complete"` (maximum/ complete linkage), `"average"` (average linkage).

`details` a Boolean determining whether intermediate logs explaining how the algorithm works should be printed or not.

`waiting` a Boolean determining whether the intermediate logs should be printed in chunks waiting for user input before printing the next or not.

`...` additional arguments passed to `proxy::dist()`.

Details

This function performs a hierarchical cluster analysis for the n objects being clustered. The definition of a set of clusters using this method follows a n step process, which repeats until a single cluster remains:

1. Initially, each object is assigned to its own cluster. The matrix of distances between clusters is computed.
2. The two clusters with closest proximity will be joined together and the proximity matrix updated. This is done according to the specified `proximity`. This step is repeated until a single cluster remains.

The definitions of `proximity` considered by this function are:

`single` $\min\{d(x,y) : x \in A, y \in B\}$. Defines the proximity between two clusters as the distance between the closest objects among the two clusters. It produces clusters where each object is closest to at least one other object in the same cluster. It is known as **SLINK**, **single-link** and **minimum-link**.

`complete` $\max\{d(x,y) : x \in A, y \in B\}$. Defines the proximity between two clusters as the distance between the furthest objects among the two clusters. It is known as **CLINK**, **complete-link** and **maximum-link**.

`average` $\frac{1}{|A| \cdot |B|} \sum_{x \in A} \sum_{y \in B} d(x,y)$. Defines the proximity between two clusters as the average distance between every pair of objects, one from each cluster. It is also known as **UPGMA** or **average-link**.

Value

An `stats::hclust()` object which describes the tree produced by the clustering process.

Author(s)

Eduardo Ruiz Sabajanes, `<eduardo.ruizs@edu.uah.es>`

Examples

```
### !! This algorithm is very slow, so we'll only test it on some datasets !!

### Helper function
test <- function(db, k, prox) {
  print(cl <- clustlearn::agglomerative_clustering(db, prox))
  oldpar <- par(mfrow = c(1, 2))
  plot(db, col = cutree(cl, k), asp = 1, pch = 20)
  h <- rev(cl$height)[50]
  clu <- as.hclust(cut(as.dendrogram(cl), h = h)$upper)
  ctr <- unique(cutree(cl, k)[cl$order])
  plot(clu, labels = FALSE, hang = -1, xlab = "Cluster", sub = "", main = "")
  rect.hclust(clu, k = k, border = ctr)
  par(oldpar)
}
```

```
### Example 1
test(clustlearn::db1, 2, "single")

### Example 2
# test(clustlearn::db2, 2, "sing") # same as "single"

### Example 3
test(clustlearn::db3, 4, "a") # same as "average"

### Example 4
test(clustlearn::db4, 6, "s") # same as "single"

### Example 5
test(clustlearn::db5, 3, "complete")

### Example 6
# test(clustlearn::db6, 3, "c") # same as "complete"

### Example 7 (with explanations, no plots)
  cl <- clustlearn::agglomerative_clustering(
  clustlearn::db5[1:6, ],
  'single',
  details = TRUE,
  waiting = FALSE
)
```

---

db1 *Test Database 1*

Description

Test Database 1

Usage

```
db1
```

Format

db1:

A data frame with 500 rows and 2 columns.

The data points form two concentric circles.

`db2` *Test Database 2*

Description

Test Database 2

Usage

```
db2
```

Format

`db2`**:**
A data frame with 500 rows and 2 columns.
The data points form two moons.

`db3` *Test Database 3*

Description

Test Database 3

Usage

```
db3
```

Format

`db3`**:**
A data frame with 500 rows and 2 columns.
The data points form three overlapping elliptical clusters of varying densities.

`db4` *Test Database 4*

Description

Test Database 4

Usage

```
db4
```

Format

`db4`**:**

A data frame with 500 rows and 2 columns.

The data points form three diagonal parallel segments.

`db5` *Test Database 5*

Description

Test Database 5

Usage

```
db5
```

Format

`db5`**:**

A data frame with 500 rows and 2 columns.

The data points form three non-overlapping circular clusters of similar density.

`db6` *Test Database 6*

Description

Test Database 6

Usage

```
db6
```

Format

`db6`**:**

A data frame with 500 rows and 2 columns.

The data points are uniformly distributed on the plane.

`dbscan` *Density Based Spatial Clustering of Applications with Noise (DBSCAN)*

Description

Perform DBSCAN clustering on a data matrix.

Usage

```
dbscan(data, eps, min_pts = 4, details = FALSE, waiting = TRUE, ...)
```

Arguments

`data` a set of observations, presented as a matrix-like object where every row is a new observation.

`eps` how close two observations have to be to be considered neighbors.

`min_pts` the minimum amount of neighbors for a region to be considered dense.

`details` a Boolean determining whether intermediate logs explaining how the algorithm works should be printed or not.

`waiting` a Boolean determining whether the intermediate logs should be printed in chunks waiting for user input before printing the next or not.

`...` additional arguments passed to `proxy::dist()`.

Details

The data given by `data` is clustered by the DBSCAN method, which aims to partition the points into clusters such that the points in a cluster are close to each other and the points in different clusters are far away from each other. The clusters are defined as dense regions of points separated by regions of low density.

The DBSCAN method follows a 2 step process:

1. For each point, the neighborhood of radius `eps` is computed. If the neighborhood contains at least `min_pts` points, then the point is considered a **core point**. Otherwise, the point is considered an **outlier**.
2. For each core point, if the core point is not already assigned to a cluster, a new cluster is created and the core point is assigned to it. Then, the neighborhood of the core point is explored. If a point in the neighborhood is a core point, then the neighborhood of that point is also explored. This process is repeated until all points in the neighborhood have been explored. If a point in the neighborhood is not already assigned to a cluster, then it is assigned to the cluster of the core point.

Whatever points are not assigned to a cluster are considered outliers.

Value

A `dbscan()` object. It is a list with the following components:

`cluster` a vector of integers (from 0 to `max(cl$cluster)`) indicating the cluster to which each point belongs. Points in clu

`eps` the value of `eps` used.

`min_pts` the value of `min_pts` used.

`size` a vector with the number of data points belonging to each cluster (where the first element is the number of outliers

Author(s)

Eduardo Ruiz Sabajanes, `<eduardo.ruizs@edu.uah.es>`

Examples

```
### Helper function
test <- function(db, eps) {
  print(cl <- clustlearn::dbscan(db, eps))
  out <- cl$cluster == 0
  plot(db[!out, ], col = cl$cluster[!out], pch = 20, asp = 1)
  points(db[out, ], col = max(cl$cluster) + 1, pch = 4, lwd = 2)
}

### Example 1
test(clustlearn::db1, 0.3)

### Example 2
test(clustlearn::db2, 0.3)

### Example 3
```

```
test(clustlearn::db3, 0.25)

### Example 4
test(clustlearn::db4, 0.2)

### Example 5
test(clustlearn::db5, 0.3)

### Example 6
test(clustlearn::db6, 0.3)

### Example 7 (with explanations, no plots)
  cl <- clustlearn::dbscan(
  clustlearn::db5[1:20, ],
  0.3,
  details = TRUE,
  waiting = FALSE
)
```

divisive_clustering *Divisive Hierarchical Clustering*

Description

Perform a hierarchical Divisive cluster analysis on a set of observations

Usage

```
divisive_clustering(data, details = FALSE, waiting = TRUE, ...)
```

Arguments

`data`	a set of observations, presented as a matrix-like object where every row is a new observation.
`details`	a Boolean determining whether intermediate logs explaining how the algorithm works should be printed or not.
`waiting`	a Boolean determining whether the intermediate logs should be printed in chunks waiting for user input before printing the next or not.
`...`	additional arguments passed to `kmeans()`.

Details

This function performs a hierarchical cluster analysis for the n objects being clustered. The definition of a set of clusters using this method follows a n step process, which repeats until n clusters remain:

1. Initially, each object is assigned to the same cluster. The sum of squares of the distances between objects and their cluster center is computed.

2. The cluster with the highest sum of squares is split into two using the k-means algorithm. This step is repeated until n clusters remain.

Value

An `stats::hclust()` object which describes the tree produced by the clustering process.

Author(s)

Eduardo Ruiz Sabajanes, <eduardo.ruizs@edu.uah.es>

Examples

```
### !! This algorithm is very slow, so we'll only test it on some datasets !!

### Helper function
test <- function(db, k) {
  print(cl <- clustlearn::divisive_clustering(db, max_iterations = 5))
  par(mfrow = c(1, 2))
  plot(db, col = cutree(cl, k), asp = 1, pch = 20)
  h <- rev(cl$height)[50]
  clu <- as.hclust(cut(as.dendrogram(cl), h = h)$upper)
  ctr <- unique(cutree(cl, k)[cl$order])
  plot(clu, labels = FALSE, hang = -1, xlab = "Cluster", sub = "", main = "")
  rect.hclust(clu, k = k, border = ctr)
}

### Example 1
# test(clustlearn::db1, 2)

### Example 2
# test(clustlearn::db2, 2)

### Example 3
# test(clustlearn::db3, 3)

### Example 4
# test(clustlearn::db4, 3)

### Example 5
test(clustlearn::db5, 3)

### Example 6
test(clustlearn::db6, 3)

### Example 7 (with explanations, no plots)
  cl <- clustlearn::divisive_clustering(
  clustlearn::db5[1:6, ],
  details = TRUE,
  waiting = FALSE
)
```

`gaussian_mixture` *Gaussian mixture model*

Description

Perform Gaussian mixture model clustering on a data matrix.

Usage

```
gaussian_mixture(data, k, max_iter = 10, details = FALSE, waiting = TRUE, ...)
```

Arguments

`data`	a set of observations, presented as a matrix-like object where every row is a new observation.
`k`	the number of clusters to find.
`max_iter`	the maximum number of iterations to perform.
`details`	a Boolean determining whether intermediate logs explaining how the algorithm works should be printed or not.
`waiting`	a Boolean determining whether the intermediate logs should be printed in chunks waiting for user input before printing the next or not.
`...`	additional arguments passed to `kmeans()`.

Details

The data given by `data` is clustered by the model-based algorithm that assumes every cluster follows a normal distribution, thus the name "Gaussian Mixture".

The normal distributions are parameterized by their mean vector, covariance matrix and mixing proportion. Initially, the mean vector is set to the cluster centers obtained by performing a k-means clustering on the data, the covariance matrix is set to the covariance matrix of the data points belonging to each cluster and the mixing proportion is set to the proportion of data points belonging to each cluster. The algorithm then optimizes the gaussian models by means of the Expectation Maximization (EM) algorithm.

The EM algorithm is an iterative algorithm that alternates between two steps:

Expectation Compute how much is each observation expected to belong to each component of the GMM.

Maximization Recompute the GMM according to the expectations from the E-step in order to maximize them.

The algorithm stops when the changes in the expectations are sufficiently small or when a maximum number of iterations is reached.

Value

A `gaussian_mixture()` object. It is a list with the following components:

cluster	a vector of integers (from 1:k) indicating the cluster to which each point belongs.
mu	the final mean parameters.
sigma	the final covariance matrices.
lambda	the final mixing proportions.
loglik	the final log likelihood.
all.loglik	a vector of each iteration's log likelihood.
iter	the number of iterations performed.
size	a vector with the number of data points belonging to each cluster.

Author(s)

Eduardo Ruiz Sabajanes, <eduardo.ruizs@edu.uah.es>

Examples

```
### !! This algorithm is very slow, so we'll only test it on some datasets !!

### Helper functions
dmnorm <- function(x, mu, sigma) {
  k <- ncol(sigma)

  x  <- as.matrix(x)
  diff <- t(t(x) - mu)

  num <- exp(-1 / 2 * diag(diff %*% solve(sigma) %*% t(diff)))
  den <- sqrt(((2 * pi)^k) * det(sigma))
  num / den
}

test <- function(db, k) {
  print(cl <- clustlearn::gaussian_mixture(db, k, 100))

  x <- seq(min(db[, 1]), max(db[, 1]), length.out = 100)
  y <- seq(min(db[, 2]), max(db[, 2]), length.out = 100)

  plot(db, col = cl$cluster, asp = 1, pch = 20)
  for (i in seq_len(k)) {
    m <- cl$mu[i, ]
    s <- cl$sigma[i, , ]
    f <- function(x, y) cl$lambda[i] * dmnorm(cbind(x, y), m, s)
    z <- outer(x, y, f)
    contour(x, y, z, col = i, add = TRUE)
  }
}

### Example 1
test(clustlearn::db1, 2)

### Example 2
# test(clustlearn::db2, 2)
```

```
### Example 3
test(clustlearn::db3, 3)

### Example 4
test(clustlearn::db4, 3)

### Example 5
test(clustlearn::db5, 3)

### Example 6
# test(clustlearn::db6, 3)

### Example 7 (with explanations, no plots)
cl <- clustlearn::gaussian_mixture(
  clustlearn::db5[1:20, ],
  3,
  details = TRUE,
  waiting = FALSE
)
```

---

kmeans *K-Means Clustering*

Description

Perform K-Means clustering on a data matrix.

Usage

```
kmeans(
  data,
  centers,
  max_iterations = 10,
  initialization = "kmeans++",
  details = FALSE,
  waiting = TRUE,
  ...
)
```

Arguments

`data`	a set of observations, presented as a matrix-like object where every row is a new observation.
`centers`	either the number of clusters or a set of initial cluster centers. If a number, the centers are chosen according to the `initialization` parameter.
`max_iterations`	the maximum number of iterations allowed.

`initialization` the initialization method to be used. This should be one of `"random"` or `"kmeans++"`. The latter is the default.

`details` a Boolean determining whether intermediate logs explaining how the algorithm works should be printed or not.

`waiting` a Boolean determining whether the intermediate logs should be printed in chunks waiting for user input before printing the next or not.

`...` additional arguments passed to `proxy::dist()`.

Details

The data given by `data` is clustered by the k-means method, which aims to partition the points into k groups such that the sum of squares from points to the assigned cluster centers is minimized. At the minimum, all cluster centers are at the mean of their Voronoi sets (the set of data points which are nearest to the cluster center).

The k-means method follows a 2 to n step process:

1. The first step can be subdivided into 3 steps:
   (a) Selection of the number k of clusters, into which the data is going to be grouped and of which the centers will be the representatives. This is determined through the use of the `centers` parameter.
   (b) Computation of the distance from each data point to each center.
   (c) Assignment of each observation to a cluster. The observation is assigned to the cluster represented by the nearest center.
2. The next steps are just like the first but for the first sub-step:
   (a) Computation of the new centers. The center of each cluster is computed as the mean of the observations assigned to said cluster.

The algorithm stops once the centers in step $n+1$ are the same as the ones in step n. However, this convergence does not always take place. For this reason, the algorithm also stops once a maximum number of iterations `max_iterations` is reached.

The `initialization` methods provided by this function are:

`random` A set of `centers` observations is chosen at random from the data as the initial centers.

`kmeans++` The `centers` observations are chosen using the **kmeans++** algorithm. This algorithm chooses the first center at random and then chooses the next center from the remaining observations with probability proportional to the square distance to the closest center. This process is repeated until `centers` centers are chosen.

Value

A `stats::kmeans()` object.

Author(s)

Eduardo Ruiz Sabajanes, `<eduardo.ruizs@edu.uah.es>`

Examples

```
### Voronoi tesselation
voronoi <- suppressMessages(suppressWarnings(require(deldir)))
cols <- c(
  "#00000019",
  "#DF536B19",
  "#61D04F19",
  "#2297E619",
  "#28E2E519",
  "#CD0BBC19",
  "#F5C71019",
  "#9E9E9E19"
)

### Helper function
test <- function(db, k) {
  print(cl <- clustlearn::kmeans(db, k, 100))
  plot(db, col = cl$cluster, asp = 1, pch = 20)
  points(cl$centers, col = seq_len(k), pch = 13, cex = 2, lwd = 2)

  if (voronoi) {
    x <- c(min(db[, 1]), max(db[, 1]))
    dx <- c(x[1] - x[2], x[2] - x[1])
    y <- c(min(db[, 2]), max(db[, 2]))
    dy <- c(y[1] - y[2], y[2] - y[1])
    tesselation <- deldir(
      cl$centers[, 1],
      cl$centers[, 2],
      rw = c(x + dx, y + dy)
    )
    tiles <- tile.list(tesselation)

    plot(
      tiles,
      asp = 1,
      add = TRUE,
      showpoints = FALSE,
      border = "#00000000",
      fillcol = cols
    )
  }
}

### Example 1
test(clustlearn::db1, 2)

### Example 2
test(clustlearn::db2, 2)

### Example 3
test(clustlearn::db3, 3)
```

```
### Example 4
test(clustlearn::db4, 3)

### Example 5
test(clustlearn::db5, 3)

### Example 6
test(clustlearn::db6, 3)

### Example 7 (with explanations, no plots)
cl <- clustlearn::kmeans(
  clustlearn::db5[1:20, ],
  3,
  details = TRUE,
  waiting = FALSE
)
```

Index