Package: UAHDataScienceUC (via r-universe)

February 17, 2025

Title Learn Clustering Techniques Through Examples and Code

Version 1.0.1

Description A comprehensive educational package combining clustering algorithms with detailed step-by-step explanations. Provides implementations of both traditional (hierarchical, k-means) and modern (Density-Based Spatial Clustering of Applications with Noise (DBSCAN), Gaussian Mixture Models (GMM), genetic k-means) clustering methods as described in Ezugwu et. al., (2022) <doi:10.1016/j.engappai.2022.104743>. Includes educational datasets highlighting different clustering challenges, based on 'scikit-learn' examples (Pedregosa et al., 2011) <`https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html`>. Features detailed algorithm explanations, visualizations, and weighted distance calculations for enhanced learning.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Depends R (>= 4.3.0)

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

Suggests deldir (>= 1.0-9), knitr, rmarkdown

VignetteBuilder knitr

LazyData true

NeedsCompilation no

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

Maintainer Andriy Protsak Protsak <`andriy.protsak@edu.uah.es`>

Repository CRAN

Date/Publication 2025-02-17 20:30:05 UTC

Contents

`agglomerative_clustering` *Agglomerative Hierarchical Clustering*

Description

Perform a hierarchical agglomerative cluster analysis on a set of observations

Usage

```
agglomerative_clustering(
  data,
  proximity = "single",
  distance_method = "euclidean",
  learn = 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).

`distance_method`

the distance measure to use. Supported values are:

- 'euclidean': Standard Euclidean distance
- 'manhattan': Manhattan (city-block) distance
- 'canberra': Canberra distance
- 'chebyshev': Chebyshev (maximum) distance

`learn` 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` $\min \left\{ d(x,y) : x \in A, y \in B \right\}$. 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` $\max \left\{ d(x,y) : x \in A, y \in B \right\}$. 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

```
cl <- agglomerative_clustering(
  db5[1:6, ],
  'single',
  learn = TRUE,
  waiting = FALSE
)
```

correlation_clustering

Hierarchical Correlation Clustering

Description

Performs hierarchical correlation clustering by applying weights, distance metrics, and other parameters to analyze relationships between data points and a target.

Usage

```
correlation_clustering(
  data,
  target = NULL,
  weight = c(),
  distance_method = "euclidean",
  normalize = TRUE,
  labels = NULL,
  learn = FALSE,
  waiting = FALSE
)
```

Arguments

`data` A data frame containing the main data

`target` A data frame, numeric vector or matrix to use as correlation target. Default is NULL.

`weight` A numeric vector of weights. Default is empty vector.

`distance_method`

A string specifying the distance metric to use. Options are:

- "euclidean" - Euclidean distance
- "manhattan" - Manhattan distance
- "canberra" - Canberra distance

- "chebyshev" - Chebyshev distance

`normalize` A boolean parameter indicating whether to normalize weights. Default is TRUE.

`labels` A string vector for graphical solution labeling. Default is NULL.

`learn` A boolean indicating whether to show detailed algorithm explanations. Default is FALSE.

`waiting` A boolean controlling pauses between explanations. Default is TRUE.

Details

This function executes the complete hierarchical correlation method in the following steps:

1. The function transforms data into useful objects
2. Creates the clusters
3. Calculates the distance from the target to every cluster using the specified distance metric
4. Orders the distances in ascending order
5. Orders the clusters according to their distance from the previous step
6. Shows the sorted clusters and the distances used

Value

An R object containing:

- dendrogram - A hierarchical clustering dendrogram
- sortedValues - A data frame with the sorted cluster values
- distances - A data frame with the sorted distances

Author(s)

Original authors:

- Roberto Alcantara <`roberto.alcantara@edu.uah.es`>
- Juan Jose Cuadrado <`jjcg@uah.es`>
- Universidad de Alcala de Henares

Examples

```
data <- matrix(c(1,2,1,4,5,1,8,2,9,6,3,5,8,5,4), ncol=3)
dataFrame <- data.frame(data)
target1 <- c(1,2,3)
target2 <- dataFrame[1,]
weight1 <- c(1,6,3)
weight2 <- c(0.1,0.6,0.3)

# Basic usage
correlation_clustering(dataFrame, target1)

# With weights
```

```
correlation_clustering(dataFrame, target1, weight1)

# Without weight normalization
correlation_clustering(dataFrame, target1, weight1, normalize = FALSE)

# Using Canberra distance with weights
correlation_clustering(dataFrame, target1, weight2, distance = "canberra", normalize = FALSE)

# With detailed explanations
correlation_clustering(dataFrame, target1, learn = TRUE)
```

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, epsilon, min_pts = 4, learn = FALSE, waiting = TRUE, ...)
```

Arguments

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

`epsilon` the value of `epsilon` 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

```
# Basic usage
data <- db5[1:20,]
result <- dbscan(data, epsilon = 0.3, min_pts = 4)

# With learning output
result <- dbscan(data, epsilon = 0.3, min_pts = 4, learn = TRUE)

# Visualize results
plot(data, col = result$cluster + 1, pch = 20)
```

`divisive_clustering` *Divisive Hierarchical Clustering*

Description

Perform a hierarchical Divisive cluster analysis on a set of observations

Usage

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

Arguments

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

`learn` 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 `clustlearn::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) {
  # Save old par settings
  old_par <- par(no.readonly = TRUE)
  # Ensure par settings are restored when function exits
  on.exit(par(old_par))

  print(cl <- 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(db1, 2)

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

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

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

```
### Example 5
test(db5, 3)

### Example 6
test(db6, 3)

### Example 7 (with explanations, no plots)
  cl <- divisive_clustering(
  db5[1:6, ],
  learn = 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, learn = 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.
learn	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 clustlearn::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 `clustlearn::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 <- 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(db1[1:250,], 2)

### Example 2 (with explanations, no plots)
cl <- gaussian_mixture(
  db5[1:10, ],
  3,
  learn = TRUE,
  waiting = FALSE
)
```

genetic_kmeans *Genetic K-Means Clustering*

Description

Performs Genetic K-Means clustering on a data matrix.

Usage

```
genetic_kmeans(
  data,
  k,
  population_size = 10,
  mut_probability = 0.5,
  max_generations = 10,
  learn = 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.

`population_size` the number of individuals in the population.

`mut_probability` the probability of a mutation occurring.

`max_generations`
: the maximum number of iterations allowed.

`learn`
: 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()`.

Value

A kmeans object as returned by the original kmeans function.

Author(s)

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

Examples

```
### Example 1: Simple usage with circles dataset
result1 <- genetic_kmeans(db1[1:20,], 2, learn = TRUE, waiting = FALSE)

### Example 2: Moons dataset with different population size
result2 <- genetic_kmeans(db2[1:20,], 2, population_size = 20,
                          learn = TRUE, waiting = FALSE)

### Example 3: Varying density clusters with different mutation probability
result3 <- genetic_kmeans(db3[1:20,], 3, mut_probability = 0.7,
                          learn = TRUE, waiting = FALSE)

### Example 4: Well-separated clusters with larger population
result5 <- genetic_kmeans(db5[1:20,], 3, population_size = 30,
                          mut_probability = 0.6, learn = TRUE, waiting = FALSE)

### Example 5: Using different parameters combinations
result6 <- genetic_kmeans(db1[1:20,], 2,
                          population_size = 15,
                          mut_probability = 0.8,
                          max_generations = 15,
                          learn = TRUE,
                          waiting = FALSE)
```

`gka_allele_mutation` *Allele mutation probability computation*

Description

Allele mutation probability computation

Usage

```
gka_allele_mutation(data, k, centers, ...)
```

Arguments

data	a set of observations, presented as a matrix-like object where every row is a new observation. The matrix is of size n by m.
k	the number of clusters.
centers	a matrix of size k by m with the cluster centers
...	additional arguments passed to proxy::dist().

Value

a matrix of size n by k with the probability of each allele mutating to a specific cluster.

Author(s)

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

---

gka_centers *Centroid computation*

Description

Centroid computation

Usage

```
gka_centers(data, k, population)
```

Arguments

data	a set of observations, presented as a matrix-like object where every row is a new observation. The matrix is of size n by m.
k	the number of clusters.
population	a matrix of size p by n with the cluster assignments for each observation.

Value

a 3D array of size p by k by m.

Author(s)

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

gka_chromosome_fix *Chromosome fixing method*

Description

This method fixes chromosomes which do not have at least one observation assigned to each cluster.

Usage

```
gka_chromosome_fix(population, k)
```

Arguments

population	a matrix of size p by n with the cluster assignments for each observation.
k	the number of clusters.

Value

a matrix of size p by n with the cluster assignments for each observation.

Author(s)

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

gka_crossover *Crossover method i.e. K-Means Operator*

Description

K-Means Operator (KMO) which replaces the crossover operator in the Genetic K-Means algorithm (GKA).

Usage

```
gka_crossover(data, centers)
```

Arguments

data	a set of observations, presented as a matrix-like object where every row is a new observation. The matrix is of size n by m.
centers	a matrix of size k by m with the cluster centers for a specific individual in the population.

Value

a vector of size n with the cluster assignments for each observation i.e. a new chromosome.

Author(s)

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

gka_fitness *Fitness function*

Description

Fitness function

Usage

```
gka_fitness(twcv)
```

Arguments

twcv	a vector of size p with the total within cluster variation of each individual in the population.

Value

a vector of size p with the fitness of each individual in the population.

Author(s)

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

gka_initialization *Initialization method*

Description

Initialization method

Usage

```
gka_initialization(n, p, k)
```

Arguments

n	the number of observations in the data.
p	the number of individuals in the population.
k	the number of clusters.

Value

a matrix of size p by n with the cluster assignments for each observation.

Author(s)

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

---

gka_mutation *Mutation method*

Description

Mutation method

Usage

```
gka_mutation(chromosome, prob, k, mut_probability)
```

Arguments

chromosome	a vector of size n with the cluster assignments for each observation.
prob	a matrix of size n by k with the probability of each allele mutating to a specific cluster.
k	the number of clusters.
mut_probability	the probability of a mutation occurring.

Value

a vector of size n with the cluster assignments for each observation i.e. a new chromosome.

Author(s)

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

gka_selection *Selection method*

Description

Selection method

Usage

```
gka_selection(p, fitness)
```

Arguments

p	the number of individuals in the population.
fitness	a vector of size p with the fitness of each individual in the population.

Value

the index of the individual selected for reproduction.

Author(s)

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

gka_twcv *Total Within Cluster Variation (TWCV) computation*

Description

Total Within Cluster Variation (TWCV) computation

Usage

```
gka_twcv(data, k, population, centers)
```

Arguments

data	a set of observations, presented as a matrix-like object where every row is a new observation. The matrix is of size n by m.
k	the number of clusters.
population	a matrix of size p by n with the cluster assignments for each observation.
centers	a 3D array of size p by k by m with the cluster centers for each individual in the population.

Value

a vector of size `p` with the total within cluster variation of each individual in the population.

Author(s)

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

`kmeans_` *K-Means Clustering*

Description

Perform K-Means clustering on a data matrix.

Usage

```
kmeans_(
  data,
  centers,
  max_iterations = 10,
  initialization = "kmeans++",
  learn = 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.

`learn` 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 <- 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(db1, 2)

### Example 2
test(db2, 2)

### Example 3
test(db3, 3)

### Example 4
test(db4, 3)

### Example 5
test(db5, 3)

### Example 6
test(db6, 3)

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

Index