Package 'qcluster'

June 5, 2026

Version 2.0.1

Date 2026-06-03

Title Clustering via Quadratic Scoring

Description Performs tuning of clustering models, methods and algorithms including the problem of determining an appropriate number of clusters. Validation of cluster analysis results is performed via quadratic scoring using resampling methods, as in Coraggio, L. and Coretto, P. (2023) <doi:10.1016/j.jmva.2023.105181>.

URL https://luca-coraggio.com, https://pietro-coretto.github.io

NeedsCompilation yes

License GPL (>= 2)

Imports cluster, doParallel, foreach, grDevices, graphics, iterators, methods, parallel, stats

Suggests RhpcBLASctl, testthat (>= 3.0.0)

LazyData TRUE

Encoding UTF-8

RoxygenNote 7.3.2

Config/testthat/edition 3

Author Luca Coraggio [cre, aut] (ORCID: <https://orcid.org/0000-0002-7898-2097>), Pietro Coretto [aut] (ORCID: <https://orcid.org/0000-0002-6972-9671>)

Maintainer Luca Coraggio <luca.coraggio@unina.it>

Repository CRAN

Date/Publication 2026-06-05 11:40:02 UTC

Contents

`banknote` *Swiss Banknotes Data*

Description

Data from Tables 1.1 and 1.2 (pp. 5-8) of Flury and Riedwyl (1988). There are six measurements made on 200 Swiss banknotes (the old-Swiss 1000-franc). The banknotes belong to two classes of equal size: *genuine* and *counterfeit*.

Format

A `data.frame` of dimension `200x7` with the following variables:

Class a `factor` with classes: `genuine`, `counterfeit`

Length Length of bill (mm)

Left Width of left edge (mm)

Right Width of right edge (mm)

Bottom Bottom margin width (mm)

Top Top margin width (mm)

Diagonal Length of diagonal (mm)

Source

Flury, B. and Riedwyl, H. (1988). *Multivariate Statistics: A practical approach.* London: Chapman & Hall.

bqs *Bootstrapping quadratic scores*

Description

Estimates the expected quadratic score for clustering solutions provided by a list of candidate models, methods or algorithmic settings.

Usage

```
bqs(
  data,
  methodset,
  B = 10,
  type = "smooth",
  oob = FALSE,
  ncores = detectCores() - 2,
  alpha = 0.05,
  rankby = ifelse(B == 0, "mean", "lq"),
  boot_na_share = 0.25,
  savescores = FALSE,
  saveparams = FALSE
)
```

Arguments

`data`	a numeric vector, matrix, or data frame of observations. Rows correspond to observations and columns correspond to variables/features. Categorical variables and `NA` values are not allowed.
`methodset`	a function, a list of functions, or a `qcmethod` object. Each method takes `data` as input and provides a clustering solution to be scored (see *Details*). See also `mset_*()` helpers and `mbind`.
`B`	an integer `>=0`. If `B=0`, methods are fitted and scored on the full data without resampling. If `B>=1`, `B` is the number of bootstrap replicates (see *Details*).
`type`	character string in `c("smooth", "hard", "both")`. `"smooth"` (default) estimates only the smooth score, `"hard"` only the hard score, and `"both"` estimates both.
`oob`	logical or character string in `c(FALSE, TRUE, "only")`. `FALSE` (default) skips out-of-bag scoring, `TRUE` adds it to the empirical bootstrap scoring, and `"only"` computes only the out-of-bag scores.
`ncores`	an integer, it defines the number of cores used for parallel computing (see *Details*). Setting `ncores = 0` uses all detected cores.
`alpha`	a number in `(0,1)`, the confidence-level for empirical bootstrap quantiles (both-tails).

rankby	character string in c("lq", "mean", "1se") or NA to skip ranking. "lq" (default) maximizes the lower limit of the 1-alpha bootstrap confidence interval, "mean" maximizes the estimated expected score, and "1se" maximizes the lower limit of a confidence interval whose semi-length is one estimated standard error.
boot_na_share	a numeric value in (0,1). During the bootstrapping a method's score is set to NA if the underlying computation runs into errors. Methods resulting in more than B * boot_na_share errors are excluded from the comparison.
savescores	logical, if =TRUE it returns estimated scores for each bootstrap sample.
saveparams	logical, if =TRUE it returns estimated cluster parameters for each bootstrap sample.

Details

The function implements the estimation and selection of an appropriate clustering based on the methodology proposed in Coraggio and Coretto (2023). It also allows out-of-bag score estimates alongside the empirical bootstrap estimates considered in that paper. Since out-of-bag estimates reuse the same bootstrap samples, oob=TRUE adds only a small extra cost. Setting B=0 gives a no-resampling baseline rather than a bootstrap estimate.

Choice of B**.** In theory B should be as large as possible, however, if the list of methods is large and the computational capacity is modest, a large B may require long run times. Coraggio and Coretto (2023) show experiments where changing from B=1000 to B=100 introduces a marginal increase in variability. B=100 should be considered as a lower bound. In the case where one has very large method lists, high-dimensional datasets and demanding methods, a possible strategy to reduce the computational cost is as follows:

1. set a small value of B, e.g., B=50 or even less.
2. Analyze the methods' ranking and identify those methods that report score values that are small compared to the top performers.
3. Narrow down the methodset list and repeat the bootstrap estimation with a value of B that is as large as possible relative to available computational resources.

Parallel computing. Bootstrap sampling is performed using foreach-based parallel computation via the doParallel parallel backend. Note that depending on the system settings, and how the functions in methodset make use of parallelism and/or multi-threading computing, increasing ncores may not produce the desired reduction in computing time. A common source of inefficiency is nested parallelism: foreach distributes work across R workers while linear algebra libraries (e.g., OpenBLAS, Intel Math Kernel Library (MKL), BLIS, Accelerate) also start their own threads inside each worker. By default, bqs avoids this oversubscription by forcing each worker to use a single BLAS/OpenMP thread whenever process-level parallelism is active.

methodset **argument.** The methodset argument allows in input a function, list, or output from mset functions: mset_user, mset_gmix, mset_kmeans, mset_pam. It is also possible to give any combination of these, concatenated with the mbind function. When passing a function, either as a single element or in a list, this must take the data set as its first argument, and must return in output at least a list named *"params"*, conforming with the return value of clust2params, i.e. a list containing proportion, mean and cov elements, representing the estimated clusters' parameters.

When rankby is not NA, the best_* components are computed by re-estimating the selected method on the full data set. For stochastic methods, use set.seed() if reproducibility is required.

Value

An S3 object of class bqs. The exact components depend on type, oob, rankby, savescores, and saveparams.

Any available score summary among smooth, hard, oob_smooth, and oob_hard is a data frame with one row per method and columns:

id index of the method in methodset;

rank rank according to rankby;

mean estimated expected score;

sterr standard error of the mean score;

lower_qnt lower confidence limit for the mean score;

upper_qnt upper confidence limit for the mean score;

n_obs number of valid bootstrap samples;

n_missing number of filtered erroneous cases.

Other output components are:

smooth returned if type="smooth" or type="both".

hard returned if type="hard" or type="both".

oob_smooth returned if type="smooth" or type="both" and oob=TRUE or oob="only".

oob_hard returned if type="hard" or type="both" and oob=TRUE or oob="only".

best_smooth list with components method_name and solution for the best smooth-scoring method. Returned only when ranking is available and a rank-1 solution exists.

best_hard analogous object for the hard score.

best_oob_smooth analogous object for the out-of-bag smooth score.

best_oob_hard analogous object for the out-of-bag hard score.

data a list containing information about the input data set necessary for the fruition of the returned object.

B number of bootstrap replicates.

methodset the elements of methodset for which a solution is produced.

rankby the ranking criterion.

raw a list returned if savescores=TRUE and/or saveparams=TRUE. Let n=sample size, B=bootstrap samples, M= number of methods in methodset. It may contain:

- boot_id: an array of dimension n x B where the j-th column contains the indexes of the observed data points belonging to the j-th bootstrap sample. That is, data[boot_id[, j],] gives the j-th bootstrap data set.
- scores: an array of dimension (M x 3 x B) returned if savescores=TRUE. It stores error codes and hard/smooth scores for each bootstrap replicate.
- oob_scores: returned if oob=TRUE or oob="only" and savescores=TRUE. It is an array organized as the previous object scores, but for out-of-bag estimates.
- params: a list returned if saveparams=TRUE. params[[m]] contains estimated cluster parameters for methodset[[m]] where m=1,...,M. Each member of the list is a list of length B where params[[m]][[b]] contains the cluster parameters fitted by the m-th method on the b-th bootstrap sample.

References

Coraggio, Luca and Pietro Coretto (2023). Selecting the number of clusters, clustering models, and algorithms. A unifying approach based on the quadratic discriminant score. *Journal of Multivariate Analysis*, Vol. 196(105181), 1-20. doi: doi:10.1016/j.jmva.2023.105181

See Also

mset_user, mset_gmix, mset_kmeans, pam, mbind, clust2params

Examples

```
# load data
data("banknote")
dat <- banknote[-1]

## set up methods
## see also help('mset_user') and related functions
KM   <- mset_kmeans(K = 3)
GMIX <- mset_gmix(K = 3, erc = c(1, 100))

# combine tuned methods
mlist <- mbind(KM, GMIX)

# perform bootstrap
# change B and ncores to a much larger value in real problems
res <- bqs(dat, mlist, B = 3, type = "both", rankby = "lq",
           ncores = 1, oob = TRUE, savescores = TRUE, saveparams = FALSE)
names(res)
res

## Not run:
# The following example is more realistic but may take time
# ----------------------------------------------------------
# load data
data("banknote")
dat <- banknote[-1]

# set up kmeans, see help('mset_kmeans')
KM <- mset_kmeans(K = 2:5)

# set up Gaussian model-based clustering via gmix()
GMIX <- mset_gmix(K = 2:5, erc = c(1, 50, 100))

# set up Gaussian model-based clustering via library("mclust")
# see examples in help('mset_user')
require(mclust)
mc_wrapper <- function(data, K, ...){
  y <- Mclust(data, G = K, ...)
  y[["params"]] <- list(proportion = y$parameters$pro,
                        mean = y$parameters$mean,
                        cov = y$parameters$variance$sigma)
  return(y)
```

```
}
MC <- mset_user(fname = "mc_wrapper", K = 2:5,
                modelNames = c("EEI", "VVV"))

# combine tuned methods
mlist <- mbind(KM, GMIX, MC)

# perform bootstrap
# set 'ncores' to the number of available physical cores
res <- bqs(dat, mlist, B = 100, type = "both", rankby = "lq", ncores = 1,
           oob = TRUE, savescores = TRUE, saveparams = FALSE)
res

## End(Not run)
```

---

bqs_rank *Ranking Clusters Quadratic Scores Estimated Via Bootstrap*

Description

Ranks the scores of clustering methods estimated via bootstrap.

Usage

```
bqs_rank(bqsol, rankby = "lq", boot_na_share = 0.25)
```

Arguments

bqsol	an object of class bqs obtained from bqs.
rankby	character string specifying how the scored solutions are ranked. Possible values are {"lq", "mean", "1se"}. With ="lq" (default), the solutions are ranked by maximizing the estimated lower limit of the 1-alpha bootstrap confidence interval for the expected score. With ="mean", the solutions are ranked by maximizing the estimated expected score. With ="1se", the solutions are ranked by maximizing the estimated lower limit of the confidence interval for the expected score whose semi-length is equal to a *standard error*. The expected score's *standard error* is approximated using the bootstrap distribution. See *Details* for small-B behavior.
boot_na_share	a numeric value in [0,1]. During the bootstrapping a method's score is set to NA if the underlying computation runs into errors. Methods resulting in more than B * boot_na_share errors are excluded from the comparison.

Details

For small B, some ranking criteria may be unstable or unavailable. In particular, with B=1, "lq" is effectively equivalent to "mean", while "1se" is not computable and the corresponding ranks remain NA. For B<5 warns that "lq" and "1se" may yield imprecise estimates.

Value

An S3 object of class bqs. Output components are those of bqs. The score-summary components are re-ranked in place. Any stored best_* components are refreshed and are returned only when a rank-1 solution exists under the new ranking. See *Value* in bqs. The object is modified only in its ranking-related components.

References

Coraggio, Luca and Pietro Coretto (2023). Selecting the number of clusters, clustering models, and algorithms. A unifying approach based on the quadratic discriminant score. *Journal of Multivariate Analysis*, Vol. 196(105181), 1-20. doi: doi:10.1016/j.jmva.2023.105181

See Also

bqs, bqs_select

Examples

```
# load data
data("banknote")
dat <- banknote[-1]

## set up methods
## see also help('mset_user') and related functions
KM   <- mset_kmeans(K = 3)
GMIX <- mset_gmix(K = 3, erc = c(1, 100))

# combine tuned methods
mlist <- mbind(KM, GMIX)

# perform bootstrap
# change B and ncores to a much larger value in real problems
res <- bqs(dat, mlist, B = 3, rankby = "lq", ncores = 1)
res

# now change ranking criterion
res2 <- bqs_rank(res, rankby = "mean")
res2
```

---

bqs_select *Select Ranked Cluster Solutions by Quadratic Score*

Description

Select solutions from a bqs object based on specified rank and type of score.

Usage

```
bqs_select(
  bqs_sol,
  rank = 1,
  type = "smooth",
  rankby = NA,
  boot_na_share = 0.25
)
```

Arguments

bqs_sol	An object of class bqs containing the clustering solutions to be selected.
rank	An integer >0 specifying the rank of the solution to select. Default is 1.
type	A character string specifying the type of Quadratic Score. Possible values are "hard", "smooth", "oob_hard", "oob_smooth". Default is "smooth".
rankby	A character string specifying the criteria used to rank solutions in bqs. Possible values are "lq", "mean", "1se", or NA (default). See *Details*.
boot_na_share	A numeric value between [0, 1]. Clustering solutions in bqs_sol with a share of NA bootstrap estimates are excluded from ranking. Default is 0.25.

Details

Even if the bqs_sol object is not pre-ranked, the user may specify a ranking criterion to rank clustering solutions dynamically using the rankby argument; this does not modify the original bqs_sol object. In these instances, the user can also specify boot_na_share as in bqs_rank to exclude solutions based on the proportion of unsuccessful bootstrap estimations. If rankby=NA, the bqs_sol must be pre-ranked.

Selected solutions are always re-estimated on the full dataset before being returned. Therefore, for stochastic clustering methods, repeated calls may return different fitted solutions unless the user controls reproducibility, e.g. via set.seed().

Value

A named list of all clustering solutions achieving a type score of rank rank when ranked according to rankby criterion, or NULL if no such solution is available in the bqs_sol object. List names correspond to methods' names in bqs_sol$methodset. Each named entry contains the corresponding method re-estimated on bqs_sol$data$data. If the full-data refit fails, the corresponding entry is an object of class bqs_select_error containing the failure status and message. If the requested rank exceeds the largest available rank, the worst available rank is returned instead. If the requested rank is within range but absent because of rank gaps, NULL is returned.

See Also

bqs, bqs_rank

Examples

```
# Load data and set seed
set.seed(123)
data("banknote")
dat <- banknote[-1]

# set up kmeans, see help('mset_kmeans')
KM <- mset_kmeans(K = 2:5)

# set up Gaussian model-based clustering via gmix()
GMIX <- mset_gmix(K = 2:5, erc = c(1, 50, 100))

# combine tuned methods
mlist <- mbind(KM, GMIX)

# perform bootstrap
# set 'ncores' to the number of available physical cores
res <- bqs(dat, mlist, B = 20, type = "both", rankby = NA, ncores = 1,
           oob = TRUE, savescores = TRUE, saveparams = FALSE)

# Methods are not ranked; this will raise an error
try(bqs_select(res, rank = 1))

# Rank method dynamically
ranked_res <- bqs_select(res, rank = 2, rankby = "lq",
                         boot_na_share = 0.25)
names(ranked_res)
```

---

clust2params *Converts Hard Assignment Into Cluster Parameters*

Description

Transforms cluster labels into a list of parameters describing cluster size, mean, and dispersion.

Usage

```
clust2params(data, cluster)
```

Arguments

data	a numeric vector, matrix, or data frame of observations. Rows correspond to observations and columns correspond to variables/features. Categorical variables and NA values are not allowed.
cluster	a vector of integers representing cluster labels. Labels need not be consecutive.

Value

A list containing cluster parameters, conformable with the Gaussian parameterization used by functions such as qscore. Let P=*number of variables/features* and K=*number of clusters*. The elements of the list are as follows:

- proportion: a vector of clusters' proportions;
- mean: a matrix of dimension (P x K) containing the clusters' mean parameters;
- cov: an array of size (P x P x K) containing the clusters' covariance matrices.

Examples

```
# load data
data("banknote")

# compute the k-means partition
set.seed(2024)
cl <- kmeans(banknote[-1], centers = 2, nstart = 1)$cluster

# convert k-means hard assignment into cluster parameters
clpars <- clust2params(banknote[-1], cl)
clpars
```

---

gmix *Gaussian Mixture Modelling*

Description

Fast implementation of the EM algorithm for ML estimation and clustering of Gaussian mixture models with covariance matrix regularization based on eigenvalue ratio constraints.

Usage

```
gmix(
  data,
  K = NA,
  erc = 50,
  iter.max = 1000,
  tol = 1e-08,
  init = "kmed",
  init.nstart = 25,
  init.iter.max = 30,
  init.tol = tol,
  save_cluster = TRUE,
  save_params = TRUE,
  save_taus = FALSE
)
```

Arguments

data	a numeric vector, matrix, or data frame of observations. Rows correspond to observations and columns correspond to variables/features. Let N=nrow(data) and P=ncol(data). Categorical variables and NA values are not allowed.
K	the number of mixture components or clusters. It can be left NA only when init is a vector of initial labels, in which case the number of clusters is retrieved from the initial partition. For character, matrix/data frame, and function initializations, K must be supplied.
erc	a numeric value >=1 specifying the eigenvalue ratio constraint (see *Details*).
iter.max	maximum number of iterations for the EM algorithm.
tol	tolerance for the convergence of the EM algorithm.
init	a character in the set c("kmed", "kmeans", "pam"), a vector, a matrix, a data frame, or a callable giving the initial assignment of data points (see *Details*). The default choice is "kmed".
init.nstart	number of initial partitions (see *Details*).
init.iter.max	maximum number of iterations for each run of the k-median initialization.
init.tol	tolerance for the convergence of each run of the k-median initialization.
save_cluster	logical, if TRUE the point-to-cluster assignment based on the *maximum a posteriori probability* (MAP) rule is returned.
save_params	logical, if TRUE the estimated mixture parameters are returned.
save_taus	logical, if TRUE the posterior class probabilities are returned (these are also known as *posterior weights* or *fuzzy weights*).

Details

The function implements the constrained ML estimator studied in Coretto and Hennig (2023). The covariance matrix constraints are computed according to the CM1-step of Algorithm 2 of Coretto and Hennig (2017). This function uses highly optimized C code for fast execution. The constrained M-step extensively uses low-level common linear algebra matrix operations (BLAS/LAPACK routines). Consequently, to maximize computational efficiency, it is recommended that the best available shared libraries, such as OpenBLAS, Intel Math Kernel Library (MKL), etc., be set up.

Initialization. The default method, set with init="kmed", uses a fast C implementation of the k-medians algorithm with random initial centers drawn uniformly over the data rows init.iter.max times. Depending on the computer power available it is suggested to set init.iter.max as large as possible particularly in cases where the data set dimensionality is large in terms of both sample size and number of features. Setting init="kmeans" replaces the k-medians with the k-means. With init="pam" initial clusters are determined using the PAM algorithm based on Euclidean distances. The latter does not perform multiple starts. The user can also set init = x where x is a vector of integers of length N=nrow(data) representing an initial hard assignment of data points to the mixture components or clusters (see *Examples*). Another possibility is to set init = W where W is a matrix or data frame of dimension (N x K) containing initial posterior probabilities or initial non-negative weights. The assignment provided via W can be hard (0-1 weights with the constraint that only a 1 is possible in each row of W) or smooth. In the current implementation, entries of W must be finite and non-negative, and each cluster must receive positive total weight. W can be seen as the initial

version of the object posterior described in the *Value* section above. The last alternative is to set init = f where f is a function with signature function(data, K) returning an N x K matrix of initial hard/smooth assignments as described for W above (see the example below).

Eigenvalue ratio constraint (erc). It is the maximum allowed ratio between within-cluster covariance matrix eigenvalues. It defines the so-called *eigenratio constraint*. erc=1 enforces spherical clusters with equal covariance matrices. A large erc allows for large between-cluster covariance discrepancies. It is suggested to never set erc arbitrarily large; its main role is to prevent degenerate covariance parameters and the related emergence of spurious clusters (see *References* below). Finally, in order to facilitate the setting of erc, it is suggested to scale the columns of data whenever measurement units of the different variables are grossly incompatible.

Value

An S3 object of class "mbcfit". Output components are as follows:

info a list with two components named code and flag giving information about the underlying EM algorithm. The code objects can take the following values:

- code=1: the algorithm converged within iter.max.
- code=2: the algorithm reached iter.max.
- code=3: the algorithm did not move from initial values.
- code=-1: unexpected memory allocation issues occurred.
- code=-2: unexpected LAPACK routine errors occurred.

The flag objects can take the following values:

- flag=0: no flag.
- flag=1: numerically degenerate posterior probabilities could not be prevented.
- flag=2: the ERC was enforced at least once.
- flag=3: conditions of flag=1 and flag=2 occurred.

iter number of iterations performed in the underlying EM algorithm.

N number of data points.

P data dimension.

K number of clusters.

loglik sample expected log-likelihood.

size cluster size (counts).

cluster cluster assignment based on the *maximum a posteriori* rule (MAP). Returned when save_cluster = TRUE.

posterior a matrix of dimension (N x K) where posterior[i, k] is the estimated posterior probability that the *i*th observation belongs to the *k*th cluster. Returned when save_taus = TRUE.

params a list containing mixture component parameters. Returned when save_params = TRUE. The elements of the list are: $proportion= vector of proportions; $mean= matrix of dimension (P x K) containing mean parameters; $cov= array of size (P x P x K) containing covariance matrices.

References

Coretto, Pietro and Christian Hennig (2017). Consistency, breakdown robustness, and algorithms for robust improper maximum likelihood clustering. *Journal of Machine Learning Research*, Vol. 18(142), pp. 1-39. URL: https://jmlr.org/papers/v18/16-382.html

Coretto, Pietro and Christian Hennig (2023) Nonparametric consistency for maximum likelihood estimation and clustering based on mixtures of elliptically-symmetric distributions. *arXiv:2311.06108*. URL: https://arxiv.org/abs/2311.06108

Examples

```
# --- load data
data("banknote")
dat <- banknote[-1]
n   <- nrow(dat) # sample size
nc  <- 2         # number of clusters

# fit 2 clusters using the default k-median initialization
set.seed(101)
fit1 <- gmix(dat, K = nc, init.nstart = 1)
print(fit1)

## Not run:
# plot partition (default)
plot(x = fit1, data = dat)

# plot partition onto the first 3 principal component coordinates
plot(x = fit1, data = prcomp(dat)$x, margins = c(1, 2, 3),
     pch_cl = c("A", "B"), col_cl = c("#4285F4", "#0F9D58"),
     main = "Principal Components")

## End(Not run)

# user-defined random initialization with hard assignment labels
set.seed(102)
i2   <- sample(1:nc, size = n, replace = TRUE)
fit2 <- gmix(dat, K = 2, init = i2)
## Not run:
plot(x = fit2, data = dat)

## End(Not run)

# user-defined smooth "toy" initialization:
# 50% of the points are assigned to cluster 1 with probability 0.9 and to
# cluster 2 with probability 0.1. The remaining data points are assigned to
# cluster 1 with probability 0.1 and to cluster 2 with probability 0.9.
set.seed(103)
idx        <- sample(c(TRUE, FALSE), size = n, replace = TRUE)
i3         <- matrix(0, nrow = n, ncol = nc)
i3[idx, ]  <- c(0.9, 0.1)
i3[!idx, ] <- c(0.1, 0.9)
fit3 <- gmix(dat, K = nc, init = i3)
```

```
## Not run:
plot(x = fit3, data = dat)

## End(Not run)

# user-defined function for initialization
# this one produces a 0-1 hard posterior matrix W based on kmeans
compute_init <- function(data, K){
  cl <- kmeans(data, K, nstart = 1, iter.max = 10)$cluster
  W  <- sapply(seq(K), function(x) as.numeric(cl == x))
  return(W)
}
fit4 <- gmix(dat, K = nc, init = compute_init)
## Not run:
plot(fit4, data = dat)

## End(Not run)
```

`mbind` *Combines Methods Settings*

Description

The function combines functions containing clustering methods setups built using `mset_user` and related functions.

Usage

```
mbind(...)
```

Arguments

`...`	one or more `qcmethod` objects obtained from `mset_user` and related functions, bare functions, or lists of bare functions. Bare functions are wrapped as unnamed user methods.

Details

`mbind()` does not modify the supplied methods; it concatenates them into a single `qcmethod` object.

Value

An S3 object of class `'qcmethod'`. Each element of the list represents a competing method containing the following objects

`fullname`	a string identifying the setup.
`callargs`	a list with arguments that are passed to the base function.

`fn` the function implementing the specified setting. This `fn` function can be executed on the data set. It has at least two arguments: `data` and `only_params`. `data` is a data matrix or data.frame `only_params` is logical. If `only_params==FALSE` (default), `fn` will return the object returned by the underlying clustering method. If `only_params==TRUE` (default) `fn` will return only cluster parameters (`proportion`, `mean`, and `cov`; see `clust2params`).

References

Coraggio, Luca and Pietro Coretto (2023). Selecting the number of clusters, clustering models, and algorithms. A unifying approach based on the quadratic discriminant score. *Journal of Multivariate Analysis*, Vol. 196(105181), 1-20. doi: doi:10.1016/j.jmva.2023.105181

See Also

`mset_user`, `mset_gmix`, `mset_kmeans`, `mset_pam`

Examples

```
# load data
data("banknote")
dat  <- banknote[-1]

# generate  kmeans setups
A <- mset_kmeans(K=c(2,3))

# generate gmix  setups
B <- mset_gmix(K=c(2,3))

# combine setups
M <- mbind(A, B)

# get one combined setting
m <- M[[4]]
m

# cluster data with 'm'
fit <- m$fn(dat)
fit
```

`mset_gmix` *Generates Methods Settings for Gaussian Mixture Model-Based Clustering*

Description

The function generates a software abstraction of a list of clustering models implemented through a set of tuned methods and algorithms. In particular, it generates a list of `gmix`-type functions each combining model tuning parameters and other algorithmic settings. The generated functions are ready to be called on the data set.

Usage

```
mset_gmix(
  K = seq(10),
  init = "kmed",
  erc = c(1, 50, 1000),
  iter.max = 1000,
  tol = 1e-08,
  init.nstart = 25,
  init.iter.max = 30,
  init.tol = tol
)
```

Arguments

K	a vector/list, specifies the number of clusters.
init	settings of the init parameter of gmix. This can be a character vector with elements in c("kmed", "kmeans", "pam"), a function, a matrix/data.frame of initial weights, or a list combining these. See gmix for the meaning of each initialization type.
erc	a vector/list, contains the settings of the erc parameter of gmix.
iter.max	a integer vector, contains the settings of the iter.max parameter of gmix.
tol	a vector/list, contains the settings of the tol parameter of gmix.
init.nstart	a integer vector, contains the settings of the init.nstart parameter of gmix.
init.iter.max	a integer vector, contains the settings of the init.iter.max parameter of gmix.
init.tol	a vector/list, contains the settings of the init.tol parameter of gmix.

Details

The function produces functions implementing competing clustering methods based on several Gaussian Mixture models specifications. The function produces functions for fitting competing Gaussian Mixture model-based clustering methods settings. This is a specialized version of the more general function mset_user. In particular, it produces a list of gmix functions each corresponding to a specific setup in terms of both model hyper-parameters (*e.g.* the number of clusters, the eigenvalue ratio constraint, *etc.*) and algorithm's control parameters (*e.g.* the type of initialization, maximum number of iteration, *etc.*). See gmix for a detailed description of the role of each argument and their data types.

Each combination of tuning parameters yields one element of the returned qcmethod object.

When init is a list, character, function, and matrix/data.frame initializations can be combined in the same qcmethod object.

Value

An S3 object of class 'qcmethod'. Each element of the list represents a competing method containing the following objects

fullname	a string identifying the setup.

callargs	a list with gmix function arguments.
fn	the function implementing the specified setting. This fn function can be executed on the data set. It has arguments data, save_cluster, save_params, save_taus, and only_params. data is a data matrix or data.frame and only_params is logical. If only_params==FALSE (default), fn will return the object returned by gmix. If only_params==TRUE, fn will return only cluster parameters (proportion, mean, and cov; see clust2params).

References

Coraggio, Luca, and Pietro Coretto (2023). Selecting the Number of Clusters, Clustering Models, and Algorithms. A Unifying Approach Based on the Quadratic Discriminant Score. *Journal of Multivariate Analysis*, Vol. 196(105181), pp. 1-20, doi:10.1016/j.jmva.2023.105181

See Also

gmix, mset_user, bqs

Examples

```
# 'gmix' settings combining number of clusters K={3,4} and eigenvalue
# ratio constraints {1,10}
A <- mset_gmix(K = c(2,3), erc = c(1,10))

# select setup 1: K=2, erc = 1, init =" kmed"
ma1 <- A[[1]]
print(ma1)

# fit A[[1]] on banknote data
data("banknote")
dat  <- banknote[-1]
fit1 <- ma1$fn(dat)
fit1

# if only cluster parameters are needed
fit1b <- ma1$fn(dat, only_params = TRUE)
fit1b


# include a custom initialization, see also help('gmix')
compute_init <- function(data, K){
  cl  <- kmeans(data, K, nstart=1, iter.max=10)$cluster
  W   <- sapply(seq(K), function(x) as.numeric(cl==x))
  return(W)
}

# generate methods settings
B <- mset_gmix(K = c(2,3), erc = c(1,10),
               init = list(compute_init, "kmed"))
```

```
# select setup 2: K=2, erc=10, init = compute_init
mb2  <- B[[2]]
fit2 <- mb2$fn(dat)
fit2
```

`mset_kmeans` *Generates Methods Settings for K-Means Clustering*

Description

The function generates a software abstraction of a list of clustering models implemented through a set of tuned methods and algorithms. In particular, it generates a list of `kmeans`-type functions each combining tuning parameters and other algorithmic settings. The generated functions are ready to be called on the data set.

Usage

```
mset_kmeans(
  K = c(1:10),
  iter.max = 50,
  nstart = 30,
  algorithm = "Hartigan-Wong",
  trace = FALSE
)
```

Arguments

`K`	a vector, specifies the number of clusters.
`iter.max`	a vector, contains the settings of the `iter.max` parameter of `kmeans`.
`nstart`	a vector, contains the settings of the `nstart` parameter of`kmeans`.
`algorithm`	a vector, contains the settings of the `algorithm` parameter of `kmeans`.
`trace`	a vector, contains the settings of the `trace` parameter of `kmeans`.

Details

The function produces functions implementing competing clustering methods based on the K-Means methodology as implemented in `kmeans`. This is a specialized version of the more general function `mset_user`. In particular, it produces a list of `kmeans` functions each corresponding to a specific setup in terms of hyper-parameters (*e.g.* the number of clusters) and algorithm's control parameters (*e.g.* initialization). See `kmeans` for a detailed description of the role of each argument and their data types.

Each combination of tuning parameters yields one element of the returned `qcmethod` object.

In the generated `fn`, the `params` component is built from the returned partition via `clust2params`.

Value

An S3 object of class 'qcmethod'. Each element of the list represents a competing method containing the following objects

fullname	a string identifying the setup.
callargs	a list with kmeans function arguments.
fn	the function implementing the specified setting. This fn function can be executed on the data set. It has two arguments: data and only_params. data is a data matrix or data.frame only_params is logical. If only_params==FALSE (default), fn will return the object returned by kmeans, augmented with a params component. If only_params==TRUE (default) fn will return only cluster parameters (proportion, mean, and cov; see clust2params).

References

Coraggio, Luca, and Pietro Coretto (2023). Selecting the Number of Clusters, Clustering Models, and Algorithms. A Unifying Approach Based on the Quadratic Discriminant Score. *Journal of Multivariate Analysis*, Vol. 196(105181), pp. 1-20, doi:10.1016/j.jmva.2023.105181

See Also

kmeans, mset_user, bqs

Examples

```
# 'kmeans' settings combining number of clusters K={2,3}
# and numbers of random starts {10,20}
A <- mset_kmeans(K = c(2,3), nstart = c(10,20))

# select setup 1: K=2, nstart = 10
m <- A[[1]]
print(m)

# cluster with the method set in 'm'
data("banknote")
dat  <- banknote[-1]
fit1 <- m$fn(dat)
fit1
class(fit1)

# if only cluster parameters are needed
fit2 <- m$fn(dat, only_params = TRUE)
fit2
```

mset_pam *Generates Methods Settings for Partitioning Around Medoids (Pam) Clustering*

Description

The function generates a software abstraction of a list of clustering models implemented through the a set of tuned methods and algorithms. In particular, it generates a list of pam-type functions each combining tuning parameters and other algorithmic settings. The generated functions are ready to be called on the data set.

Usage

```
mset_pam(
  K = seq(10),
  metric = "euclidean",
  medoids = if (is.numeric(nstart)) "random",
  nstart = if (variant == "faster") 1 else NA,
  stand = FALSE,
  do.swap = TRUE,
  variant = "original",
  pamonce = FALSE
)
```

Arguments

K	a vector/list, specifies the number of clusters.
metric	a vector, contains the settings of the metric parameter of pam.
medoids	settings of the medoids parameter of pam. This can be a character vector, a numeric vector of user-supplied medoid labels, or a list combining these.
nstart	a vector, contains the settings of the nstart parameter of pam.
stand	a vector, contains the settings of the stand parameter of pam.
do.swap	a vector, contains the settings of the do.swap parameter of pam.
variant	a list, contains the settings of the variant parameter of pam.
pamonce	a vector, contains the settings of the pamonce parameter of pam.

Details

The function produces functions implementing competing clustering methods based on the PAM clustering methodology as implemented in pam. This is a specialized version of the more general function mset_user. In particular, it produces a list of pam functions each corresponding to a specific setup in terms of hyper-parameters (*e.g.* the number of clusters) and algorithm's control parameters (*e.g.* initialization). See pam for a detailed description of the role of each argument and their data types.

Each combination of tuning parameters yields one element of the returned qcmethod object.

When `medoids` is numeric or a list containing numeric entries, the corresponding number of clusters is derived from the supplied labels.

In the generated `fn`, the `params` component is built from the returned partition via `clust2params`.

Value

An S3 object of class `'qcmethod'`. Each element of the list represents a competing method containing the following objects

`fullname`	a string identifying the setup.
`callargs`	a list with `pam` function arguments.
`fn`	the function implementing the specified setting. This `fn` function can be executed on the data set. It has arguments `data`, `diss`, `cluster.only`, `keep.data`, `keep.diss`, and `only_params`. `data` is a data matrix or data.frame and `only_params` is logical. If `only_params==FALSE` (default), `fn` will return the object returned by `pam`, augmented with a `params` component. If `only_params==TRUE` (default) `fn` will return only cluster parameters (`proportion`, `mean`, and `cov`; see `clust2params`).

References

Coraggio, Luca, and Pietro Coretto (2023). Selecting the Number of Clusters, Clustering Models, and Algorithms. A Unifying Approach Based on the Quadratic Discriminant Score. *Journal of Multivariate Analysis*, Vol. 196(105181), pp. 1-20, doi:10.1016/j.jmva.2023.105181

See Also

`pam`,`mset_user`, `bqs`

Examples

```
# 'pam' settings combining number of clusters K={2,3}, and dissimilarities {euclidean, manhattan}
A <- mset_pam(K = c(2,3), metric = c("euclidean", "manhattan"))

# select setup 1: K=2, metric = "euclidean"
m <- A[[1]]
print(m)


# cluster with the method set in 'm'
data("banknote")
dat  <- banknote[-1]
fit1 <- m$fn(dat)
fit1
class(fit1)


# if only cluster parameters are needed
fit1b <- m$fn(dat, only_params = TRUE)
fit1b
```

mset_user *Generates Clustering Methods Settings for a Prototype Methodology Provided by the User*

Description

The function generates a software abstraction of a list of clustering models implemented through the a set of tuned methods and algorithms. The *base* clustering methodology is provided via a user-defined function. The latter prototype is exapanded in a list of fucntions each combining tuning parameters and other algorithmic settings. The generated functions are ready to be called on the data set.

Usage

```
mset_user(fname, .packages = NULL, .export = NULL, ...)
```

Arguments

fname	the name of a function implementing a user-defined clustering method. It clusters a data set and outputs cluster parameters. fname must fulfill certain requirements detailed below in the *Details*.
.packages	character vector of packages that the tasks in fname depend on (see *Details*).
.export	character vector of variables to export that are needed by fname and that are not defined in the current environment (see *Details*).
...	parameters passed to fname. If a given parameter is included as a vector/list each of its members is to obtain the final collection of fname specifications (see *Details* and *Examples*).

Details

The function produces functions implementing competing clustering methods based on a *prototype* methodology implemented by the user via the input argument fname. In particular, it builds a list of fname-type functions each corresponding to a specific setup in terms of hyper-parameters (*e.g.* the number of clusters) and algorithm's control parameters (*e.g.* initialization).

Each combination of tuning parameters yields one element of the returned qcmethod object.

Requirements for fname. fname must be the name of a callable function implementing the base clustering method of interest. It must have the following input argument

- data: a numeric vector, matrix, or data frame of observations. Rows correspond to observations and columns correspond to variables/features. Categorical variables and NA values are not allowed.

Additionally, fname can have any other input parameter controlling the underlying clustering model/method/algorithm. All this additional parameters are passed to mset_user via ... (see *Arguments*).

The output of fname must contain a list named params with cluster parameters describing size, centrality and scatter. Let P= *number of variable/features* and K= *number of clusters*. The elements of params are as follows:

- proportion: a vector of clusters' proportions;
- mean: a matrix of dimension (P x K) containing the clusters' mean parameters;
- cov: an array of size (P x P x K) containing the clusters' covariance matrices.

Note that params can be easily obtained from a vector of cluster labels using clust2params.

.packages **and** .export**.** The user does not normally need to specify .packages and .export. These arguments are not needed if the functions generated by mset_user will be called from an environment containing all variables and functions needed to execute fname. Functions like bqs will call the functions generated by mset_user within a parallel infrastructure using foreach. If the user specifies .packages and .export, they will be passed to the .packages and .export arguments of foreach.

The generated fn returns res$params unchanged when only_params = TRUE.

Finally, note that the package already contains specialized versions of mset_user generating methods settings for some popular algorithms (see mset_gmix, mset_kmeans, mset_pam)

Value

An S3 object of class 'qcmethod'. Each element of the list represents a competing method containing the following objects

fullname	a string identifying the setup.
callargs	a list with arguments that are passed to the base function.
fn	the function implementing the specified setting. This fn function can be executed on the data set. It has two arguments: data and only_params. data is a data matrix or data.frame only_params is logical. If only_params==FALSE (default), fn will return the object returned by fname. If only_params==TRUE (default) fn will return only cluster parameters (proportion, mean, and cov; see clust2params).

References

Coraggio, Luca, and Pietro Coretto (2023). Selecting the Number of Clusters, Clustering Models, and Algorithms. A Unifying Approach Based on the Quadratic Discriminant Score. *Journal of Multivariate Analysis*, Vol. 196(105181), pp. 1-20, doi:10.1016/j.jmva.2023.105181

See Also

clust2params, mset_gmix, mset_kmeans, mset_pam

Examples

```
# load data
data("banknote")
dat  <- banknote[-1]

# EXAMPLE 1: generate Hierarchical Clustering settings
# ----------------------------------------------------
```

```
# wrapper for the popular stats::hclust() for Hierarchical Clustering
# Note the usee:
#   of the optional arguments '...' passed to the underling clustering function
#   the use of 'clust2params' to add cluster parameters to the output
hc_wrapper <- function(data, K, ...){
    dm  <- dist(data, method = "euclidean")
    ## ... = hc parameters
    hc  <- hclust(dm, ...)
    cl  <- cutree(hc, k = K)
    ## output with params
    res          <- list()
    res$cluster  <- cl
    res$params   <- clust2params(data, cluster = cl)
    return(res)
}

# generate settings for Hierarchical Clustering with varying
# number of clusters K={3,4},  agglomeration method = {ward.D, median}
# see help('stats::hclust')
A <- mset_user(fname="hc_wrapper", K = c(2,3), method = c("ward.D", "complete"))

# get the setting with K=2 and method = "complete"
ma <- A[[4]]
ma

# cluster data with 'ma'
fit_a1 <- ma$fn(dat)
fit_a1

## if only cluster parameters are needed
fit_a2 <- ma$fn(dat, only_params = TRUE)
fit_a2


## Not run:
# EXAMPLE 2: generate 'mclust' model settings
# ------------------------------------------
# mclust is popular package for performing model based clustering based on
# Gaussian mixture. Please visit
# https://cran.r-project.org/web/packages/mclust/vignettes/mclust.html
require(mclust)

# wrapper for the popular stats::hclust() for Hierarchical Clustering
# Notes:
#  * optional arguments '...' are passed to the underling
#    'mclust' clustering function
#  * 'mclust' fits Gaussian Mixture models so cluster parameters are
#     contained in the mclust object
mc_wrapper <- function(data, K, ...){
    y <- Mclust(data, G = K, ...)
    y[["params"]] <- list(proportion = y$parameters$pro,
                          mean = y$parameters$mean,
                          cov = y$parameters$variance$sigma)
```

```
    return(y)
    }

# generate 'mclust' model settings by varying the number of clusters and
# covariance matrix models (see help('mclust::mclustModelNames'))
B <- mset_user(fname = "mc_wrapper", K = c(2,3), modelNames = c("EEI", "VVV"))


# get the setting with K=3 and covariance model "EEI"
mb <- B[[2]]
mb

# cluster data with 'mb'
fit_b <- mb$fn(dat)
fit_b ## class(fit_b) = "Mclust"


# if needed one can make sure that 'mclust' package is always available
# by setting the argument '.packages'
B <- mset_user(fname = "mc_wrapper", K = c(2,3), modelNames = c("EEI","VVV"),
               .packages = c("mclust"))

## End(Not run)

## Not run:
# EXAMPLE 3: generate 'dbscan' settings
# -------------------------------------
# DBSCAN is popular nonparametric method for discovering clusters of
# arbitrary shapes with noise. The number of clusters is implicitly
# determined via two crucial tunings usually called 'eps' and 'minPts'
# See https://en.wikipedia.org/wiki/DBSCAN
require(dbscan)

# wrapper for dbscan::dbscan
db_wrap <- function(data, ...) {
  cl <- dbscan(data, borderPoints = TRUE, ...)$cluster
  return(list(params = clust2params(data, cl)))
}

D  <- mset_user(fname = "db_wrap", eps = c(0.5, 1), minPts=c(5,10))
md    <- D[[2]]
fit_d <- md$fn(dat)
fit_d
class(fit_d)

## End(Not run)
```

---

plot.bqs *Plot (Bootstrap) Quadratic Score Results*

Description

Produce a plot of bqs (Bootstrap Quadratic Scores). This function creates plots based on the BQS (Bootstrap Quality Scores) data.

Usage

```
## S3 method for class 'bqs'
plot(x, score = NULL, perc_scale = FALSE, top = NULL, annotate = NULL, ...)
```

Arguments

x	An S3 object of class bqs as returned by the bqs function. x must be ranked, i.e. it must have a non-missing rankby component.
score	Character vector specifying the score(s) to be plotted. Valid scores are "hard", "smooth", "oob_hard", and "oob_smooth". If NULL (default), all score components available in x are plotted.
perc_scale	Logical; if TRUE, scales the plot using percentages, relative to the best score. Default is FALSE.
top	Numeric; specifies the number of top models to individually highlight. Must be a single number less than or equal to the length of x$methodset. If NULL (default), top is automatically determined based on the score values. Top models are emphasized visually, not filtered.
annotate	Logical; if TRUE, annotates the top models in the plot. Default is automatically determined (TRUE if the number of methods M <= 30, FALSE otherwise). Annotations include method names and plotted score values.
...	Further arguments passed to or from other methods.

Details

The annotate argument is mainly intended to control built-in labels, but setting it to FALSE also leaves more room for expert users to add custom annotations after the plot is drawn.

Value

No return value, called for side effects.

See Also

bqs

Examples

```
# load data
data("banknote")
dat <- banknote[-1]

# set up methods
mlist <- mset_gmix(K = 1:3, erc = c(1, 100))
```

```
# perform bootstrap
# change B and ncores to a much larger value in real problems
res <- bqs(dat, mlist, B = 3, type = "both", rankby = "lq",
           ncores = 1, oob = TRUE, savescores = FALSE, saveparams = FALSE)

# Plot with default settings
plot(res)

# Plot in percentage scale relative to first model
plot(res, perc_scale = TRUE)
```

`plot.mbcfit` *Plot Fitted Mixture Models*

Description

This function provides a plot method for objects of class `mbcfit`, returned as output by the `gmix` function. It serves as a wrapper around `plot_clustering`, allowing easy visualization of clustering results, including clustering assignments, contours, and boundaries.

Usage

```
## S3 method for class 'mbcfit'
plot(
  x,
  data = NULL,
  subset = NULL,
  what = c("clustering", "contour"),
  col_cl = NULL,
  pch_cl = NULL,
  ...
)
```

Arguments

`x`	An object of class `mbcfit`, typically a result of the `gmix` function.
`data`	`NULL` or a data matrix, data frame, or array containing data points to be plotted. When supplied, it is typically the data set used to fit `x`. See *Details*.
`subset`	A numeric vector indexing columns of `data` to subset and focus the plot on specific features. Default is `NULL`.
`what`	Character vector specifying which elements to plot. Options are `"clustering"`, `"contour"`, and `"boundary"`. Default is to plot `"clustering"` and `"contour"`. Unavailable plot features are ignored or rejected downstream by `plot_clustering` depending on the inputs. See *Details*.

col_cl	A vector of colors to use for clusters (one for each cluster). Default is NULL, which uses a default sequence of colors.
pch_cl	A vector of plotting symbols (one for each cluster) to use for clusters. Default is NULL, which uses a default sequence of symbols.
...	Further arguments passed to or from other methods.

Details

The plot.mbcfit function provides a plotting method for objects of class mbcfit. It acts as a wrapper around the plot_clustering function, allowing users to easily generate various plots to analyze the clustering results. A plot is produced only upon a successful mbcfit estimate, i.e., when mbcfit has code equal to either 1 or 2. The plot features that can actually be drawn depend on the components stored in x, in particular x$params and x$cluster.

When data is NULL (the default), the function constructs a synthetic plotting data set from the fitted params and then delegates the plotting to plot_clustering.

When data is not NULL, the function passes data, the stored mixture parameters, and the hard clustering labels saved in x$cluster to plot_clustering.

Value

Called for its side effects.

See Also

gmix, plot_clustering, predict.mbcfit

Examples

```
# load data
data("banknote")
dat <- banknote[-1]

# fit 2 clusters
set.seed(123)
fit <- gmix(dat, K = 2, init.nstart = 1)
print(fit)

# plot partition (default)
plot(x = fit, data = dat)


# plot partition onto the first 3 coordinates
plot(x = fit, data = dat, subset = c(1:3), pch_cl = c("A", "B"),
     col_cl = c("#4285F4", "#0F9D58"), what = "clustering")

# additionally plot clustering boundary and contour sets
plot(x = fit, data = dat, subset = c(1:3), pch_cl = c("A", "B"),
     col_cl = c("#4285F4", "#0F9D58"),
     what = c("clustering", "boundary", "contour"))
```

plot_clustering *Plot Data With Clustering Information*

Description

This function plots data and optionally adds clustering information such as clustering assignments, contours, or boundaries.

Usage

```
plot_clustering(
  data,
  subset = NULL,
  cluster = NULL,
  params = NULL,
  what = c("clustering", "contour", "boundary"),
  col_cl = NULL,
  pch_cl = NULL
)
```

Arguments

data	a numeric vector, matrix, or data frame of observations. Rows correspond to observations and columns correspond to variables/features. Categorical variables and NA values are not allowed.
subset	A numeric vector indexing columns of data to be visualized. Default is NULL.
cluster	A vector of cluster assignments. If provided, the plot can display clustering information as specified in what. Must have the same number of observations as data.
params	A list of clustering parameters, including proportion, mean, and cov. If provided, the plot can display contour and boundary information. For "contour" and "boundary", params must be valid mixture parameters.
what	Character vector specifying which elements to plot. Options are "clustering", "contour", and "boundary". The default request is all three. Requested features that are incompatible with the supplied inputs are dropped with a warning; if none remain, the function stops with an error.
col_cl	A vector of colors to use for clusters (one for each cluster). Default is NULL, which uses a default sequence of colors.
pch_cl	A vector of plotting symbols (one for each cluster) to use for clusters. Default is NULL, which uses a default sequence of symbols.

Details

At least one of `cluster` or `params` must be supplied. Contours and boundaries require Gaussian-ready parameters. Depending on the data dimension, the function produces one-dimensional plots, two-dimensional plots, or a scatterplot matrix over feature pairs. When `subset` is used, `params` are restricted to the selected coordinates before plotting.

Value

No return value, called for side effects.

See Also

`bqs`, `clust2params`

Examples

```
# Example data
set.seed(123)
data <- rbind(
            matrix(rnorm(100 * 2), ncol = 2),
            matrix(rnorm(100 * 2) + 2, ncol = 2)
        )
cluster <- c(rep(1, 100), rep(2, 100))
params <- clust2params(data, cluster)

# Plot with clustering information
plot_clustering(data, cluster = cluster, what = "clustering")

# Plot with subset of variables
plot_clustering(data, cluster = cluster, subset = 1,
                what = c("clustering", "contour"))

# Plot with customized colors and symbols
plot_clustering(data, cluster = cluster, params = params,
                col_cl = c("magenta", "orange"),
                pch_cl = c("A", "B"))
```

`predict.mbcfit` *Predict Hard Clustering Assignments for using Mixture Models*

Description

This function predicts cluster assignments for new data based on an existing model of class `mbcfit`. The prediction leverages information from the fitted model to categorize new observations into clusters.

Usage

```
## S3 method for class 'mbcfit'
predict(object, newdata, ...)
```

Arguments

`object`	An object of class `mbcfit`, representing the fitted mixture model. This is typically obtained in output from the `gmix` function. See *Details*.
`newdata`	A numeric vector, matrix, or data frame of observations. Rows correspond to observations and columns correspond to variables/features. Categorical variables and `NA` values are not allowed. The number of columns must be coherent with that implied by `object`. See *Details*.
`...`	Further arguments passed to or from other methods.

Details

The `predict.mbcfit` function utilizes the parameters of a previously fitted `mbcfit` model to allocate new data points to estimated clusters. The function performs necessary checks to ensure the `mbcfit` model returns valid estimates and the dimensionality of the new data aligns with the model.

The `mbcfit` object must contain a component named `params`, which is itself a list containing the following necessary elements, for a mixture model with K components:

`proportion` A numeric vector of length K, with elements summing to 1, representing cluster proportions.

`mean` A numeric matrix of dimensions `c(P, K)`, representing cluster centers.

`cov` A numeric array of dimensions `c(P, P, K)`, representing cluster covariance matrices.

Data dimensionality is `P`, and new data dimensionality must match (`ncol(newdata)` must be equal to `P`) or otherwise the function terminates with an error message. The stored mixture parameters must be Gaussian-ready, i.e. proportions must be positive and sum to 1, and covariance matrices must be finite, symmetric, and positive definite.

The predicted clustering is obtained as the MAP estimator using posterior weights of a Gaussian mixture model parametrized at `params`. Denoting with $z(x)$ the predicted cluster label for point x, and with ϕ the (multivariate) Gaussian density:

$$z(x) = \underset{k=\{1,\dots,K\}}{\arg\max} \frac{\pi_k \phi(x, \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \phi(x, \mu_j, \Sigma_j)}$$

Value

A vector of predicted cluster labels, one for each observation in `newdata`.

References

Coraggio, Luca and Pietro Coretto (2023). Selecting the number of clusters, clustering models, and algorithms. A unifying approach based on the quadratic discriminant score. *Journal of Multivariate Analysis*, Vol. 196(105181), 1-20. doi: doi:10.1016/j.jmva.2023.105181

See Also

gmix

Examples

```
# load data
data(banknote)
dat <- banknote[, -1]

# Estimate 3-components gaussian mixture model
set.seed(123)
res <- gmix(dat, K = 3)

# Cluster in output from gmix
print(res$cluster)

# Predict cluster on a single point
# (keep table dimension)
predict(res, dat[1, , drop = FALSE])

# Predict cluster on a subset
predict(res, dat[1:10, ])

# Predicted cluster on original dataset are equal to the clustering from the
# gmix model
all(predict(res, dat) == res$cluster)
```

---

print.bqs *Display Information on Bootstrap Quadratic Scores Objects*

Description

This function provides a print method for objects of class bqs, which are produced by the bqs function. It prints a summary of the bootstrapped quadratic score results for the clustering solutions considered.

Usage

```
## S3 method for class 'bqs'
print(x, ...)
```

Arguments

x	An object of class bqs, usually the output of the bqs function.
...	Additional arguments passed to or from other methods.

Details

The `print.bqs` function provides a print method for objects of class `bqs`.

If clustering solutions in `bqs` are not ranked, the printing method displays a message to the user signalling it. Otherwise, the printing method shows a summary of the top-6 ranked solutions, in increasing rank order, for any available scoring method. The available scoring methods are determined by the `type` and `oob` arguments used in input to the `bqs` function (see Details in `bqs`).

The summary tables for ranked methods has `row.names` set to the method's codename, and shows the following information along the columns:

`id` Method's index in the `methodset` list (see Details in `bqs`).

`rank` Method's rank according to ranking criterion.

`score` (Only shown when `B=0`) Method's quadratic score on the full data.

`mean` (Only shown when `B>0`) Method's mean bootstrap quadratic score.

`sterr` (Only shown when `B>0`) Method's standard error for the bootstrap quadratic score.

`lower_qnt` (Only shown for `"mean"` and `"lq"` ranking, when `B>0`) Method's lower `alpha/2`-level quantile of the bootstrap distribution of the quadratic score (`alpha` is given in input to `bqs` function).

`upper_qnt` (Only shown for `"mean"` and `"lq"` ranking, when `B>0`) Method's upper `alpha/2`-level quantile of the bootstrap distribution of the quadratic score (`alpha` is given in input to `bqs` function).

`-1se` (Only shown for `"1se"` ranking, when `B>0`) Method's mean bootstrap quadratic score minus 1 standard error.

`+1se` (Only shown for `"1se"` ranking, when `B>0`) Method's mean bootstrap quadratic score plus 1 standard error.

Methods with missing ranks are omitted from the printed summary. If all ranks are missing for a given score component, a short message is printed instead of a table.

Value

No return value, called for side effects.

See Also

`bqs`, `bqs_rank`

Examples

```
# Load data and set seet
set.seed(123)
data("banknote")
dat <- banknote[-1]

# set up kmeans, see help('mset_kmeans')
KM    <- mset_kmeans(K = 2:5)

# set up Gaussian model-based clustering via gmix()
```

```
GMIX  <- mset_gmix(K=2:5, erc=c(1, 50 , 100))

# combine tuned methods
mlist <- mbind(KM, GMIX)

# perform bootstrap
# se 'ncores' to the number of available physical cores
res <- bqs(dat, mlist, B = 100, type = "both", rankby=NA, ncores = 1,
           oob = TRUE, savescores = TRUE, saveparams = FALSE)

# Methods are not ranked; only available components are shown
res

# Rank method and show summaries
ranked_res <- bqs_rank(res, rankby = "lq", boot_na_share = 0.25)

ranked_res
```

`print.mbcfit` *Display Information for Mixture Model Objects*

Description

This function provides a print method for objects of class `mbcfit`, returned in output by the `gmix` function.

Usage

```
## S3 method for class 'mbcfit'
print(x, ...)
```

Arguments

`x`	An object of class `mbcfit`, typically a result of the `gmix` function.
`...`	Further arguments passed to or from other methods.

Details

The `print.mbcfit` function gives a summary of a model-based clustering fit, estimated using the `gmix` function.

The printed message depends on `x$info$code`:

- `-2 'Lapack DSYEV failed'`. This error occurs whenever any of the cluster-covariance matrices becomes singular during estimation, using the EM algorithm.
- `-1 'Memory allocation error'`. This error occurs when there is insufficient available memory to allocate the quantities required to execute the EM algorithm.

1 Success.

2 'gmix' did not converge (iterations reached the maximum limit).

3 EM algorithm failed; no better than the initial solution. This error occurs whenever the EM algorithm failed for other reasons (e.g., degenerate posterior-weights could not be prevented), and it was not possible to find a solution.

The printed output also lists available components of the mbcfit object and summarizes the number of clusters found and their size, whenever this information is available.

Value

No return value, called for side effects.

See Also

gmix

Examples

```
set.seed(123)

# Estimate a 3-clusters Gaussian mixture model, using iris data as example
res <- gmix(iris[, -5], K = 3, erc = 10)

# Print the 'gmix' output
print(res)
```

---

qcluster *qcluster: Clustering via Quadratic Scoring*

Description

qcluster provides tools for tuning clustering models, methods, and algorithms by quadratic scoring with resampling.

Details

The package combines three main components:

- method generators such as mset_user, mset_gmix, mset_kmeans, mset_pam, and mbind;
- bootstrap quadratic-score estimation, ranking, and selection through bqs, bqs_rank, and bqs_select;
- direct Gaussian model-based clustering and scoring via gmix and qscore.

A typical workflow is:

1. define a collection of candidate clustering methods with mset_*() and optionally combine them with mbind;

2. estimate bootstrap quadratic scores with bqs;
3. rank candidate methods with bqs_rank and extract selected full-data refits with bqs_select.

The package also provides plotting and printing methods for fitted mbcfit and bqs objects, together with the sample data set banknote.

Author(s)

Maintainer: Luca Coraggio <luca.coraggio@unina.it> (ORCID)

Authors:

- Pietro Coretto <pcoretto@unisa.it> (ORCID)

References

Coraggio, Luca and Pietro Coretto (2023). Selecting the number of clusters, clustering models, and algorithms. A unifying approach based on the quadratic discriminant score. *Journal of Multivariate Analysis*, Vol. 196(105181), 1-20. doi: doi:10.1016/j.jmva.2023.105181

See Also

Useful links:

- https://luca-coraggio.com
- https://pietro-coretto.github.io

---

qscore *Clustering Quadratic Score*

Description

Computes both the hard and the smooth quadratic score of a clustering.

Usage

```
qscore(data, params, type = "both")
```

Arguments

data	a numeric vector, matrix, or data frame of observations. Rows correspond to observations and columns correspond to variables/features. Let N=nrow(data) and P=ncol(data). Categorical variables and NA values are not allowed.
params	a list containing cluster parameters *(proportion, mean, cov)*. Let K=*number of clusters*. The elements of the list are as follows: $proportion= vector of clusters' proportions; $mean= matrix of dimension (P x K) containing the clusters' mean parameters; $cov= array of size (P x P x K) containing the clusters' covariance matrices. These parameters must be conformable with data; see also clust2params.
type	the type of score, a character in the set c("both", "smooth", "hard"). The default value is set to "both". See *Details*.

Details

The function calculates quadratic scores as defined in equation (22) in Coraggio and Coretto (2023). The score is computed from a Gaussian parameterization supplied in `params`.

Value

A named numeric vector with components `hard` and `smooth`. When only one score is requested through `type`, the other component is returned as `NA`.

References

Coraggio, Luca and Pietro Coretto (2023). Selecting the number of clusters, clustering models, and algorithms. A unifying approach based on the quadratic discriminant score. *Journal of Multivariate Analysis*, Vol. 196(105181), 1-20. DOI: doi:10.1016/j.jmva.2023.105181

See Also

`clust2params`, `gmix`

Examples

```
# --- load and split data
data("banknote")
set.seed(345)
idx   <- sample(1:nrow(banknote), size = 25, replace = FALSE)
dat_f <- banknote[-idx, -1] ## training data set
dat_v <- banknote[ idx, -1] ## validation data set

# --- Gaussian model-based clustering, K=3
# fit clusters
fit1 <- gmix(dat_f, K = 3)
## compute quadratic scores using fitted mixture parameters
s1 <- qscore(dat_v, params = fit1$params)
s1

# --- k-means clustering, K=3
# obtain the k-means partition
cl_km <- kmeans(dat_f, centers = 3, nstart = 1)$cluster
## convert k-means hard assignment into cluster parameters
par_km <- clust2params(dat_f, cl_km)
# compute quadratic scores
s2 <- qscore(dat_v, params = par_km)
s2
```

Index