Package 'qcluster'

July 22, 2025

Version 1.2.1

Date 2024-12-23

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>.

NeedsCompilation yes

License GPL (>= 2)

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

LazyData TRUE

Encoding UTF-8

RoxygenNote 7.3.2

Author Luca Coraggio [cre, aut] (Homepage: <https://luca-coraggio.com>), Pietro Coretto [aut] (Homepage: <https://pietro-coretto.github.io>)

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

Repository CRAN

Date/Publication 2025-01-13 19:10:02 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*.

Usage

```
data(banknote)
```

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 of candidate model, methods or algorithmic setting.

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 list of functions. A function in the list takes data as input and provide a clustering method to be scored (see *Details*).
B	a integer >=0. If B=0, the funciton fits and scores the clustering methods on the entire data set without resampling (see *Details*). B>=1, sets the number of boostrap replicates (see *Details*).
type	character string specifying the type of score, Possibile values are {"smooth", "hard", "both"}. If ="smooth" (default), only the smooth score is estimated. If ="hard", only the hard score is estimated. ="both", both the smooth and the hard scores are estimated.
oob	logical or character string specifying if out-of-bag bootstrap is performed. Possibile values are {FALSE, TRUE, "only"} If =FALSE (default), out-of-bag boostrap is not performed. If =TRUE, out-of-bag bootstrap is performed along with the empirical bootstrap sampling. If ="only", only the out-of-bag bootstrap is performed.
ncores	an integer, it defines the number of cores used for parallel computing (see *Details*).
alpha	a number in (0,1), the confidence-level for empirical bootstrap quantiles (both-tails).

rankby	character string specifying how the scored solution are ranked. Possible values are {"lq", "mean", "1se"}. With ="lq" (default), the solutions are ranked by maximizing the estimated lower limit of the of the 1-alpha bootstrap confidence intervalfor 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 boostrap distribution.
boot_na_share	a numeric value in (0,1). During the boostrapping a method's score is set to NA if the underlying comptutation 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 boostrap sample.
saveparams	logical, if =TRUE it returns estimated cluster parameters for each boostrap sample.

Details

The function implements the estimation and selection of an appropriate clustering based on the methodology proposed in Coraggio and Coretto (2023). In addition, we add the possibility of obtaining score estimates using out-of-bag-boorstrap sampling alongside the empirical bootstrap-based estimates proposed in the aforementioned paper. Note that the out-of-bag-boorstrap estimates are obtained using the same samples used for the emprical bootsrap, therefore, oob=TRUE add a small computational cost.

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. For instance, this happens when using linear algebra routines optimized for multi-threaded computing (e.g., OpenBLAS, Intel Math Kernel Library (MKL), and so on). These optimized shared libraries already implement multi-threading, and it is necessary to find the optimal trade-off between distributing processes over physical cores and multi-threading at the logical unit level of the same physical unit. Unfortunately, there is no universal recipe and much depends on hardware architectures, operating system, shared libraries, etc. We obtained the best results using OpenBLAS (the tagged *serial*) and setting ncores= the number of physical cores.

`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 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 `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.

Value

An S3 object of class `bqs`. Output components are as follows:

`smooth` data.frame returned if `type="smooth"` or `type="both"`. It contains a summary of the estimated score. The rows corresponds to competing methods in `methodset` sorted by the specified ranking criterion. Columns are as follows:

- `id:` index of the method in the corresponding `methodset` list;
- `rank:` rank of the clustering solution according to `rankby`;
- `mean:` expected score;
- `sterr` standard error for the mean score;
- `lower_qnt:` lower limit of the confidence interval for the mean score;
- `upper_qnt:` upper limit of the confidence interval for the mean score;
- `n_obs:` number of valid bootstrap samples after filtering erroneous cases;
- `n_missing:` number of filtered erroneous cases.

`hard` data.frame returned if `type="hard"` or `type="both"`. It reports the results about the hard score in analogy to the previous object `smooth`.

`obb_smooth` data.frame returned if `type="smooth"` or `type="both"` and `obb=TRUE` or `obb="only"`. It reports the results about the smooth score estimated using out-of-bang-boostrap samples in analogy to the previous objects `smooth` and `hard`.

`obb_hard` data.frame returned if `type="hard"` or `type="both"` and `obb=TRUE` or `obb="only"`. It reports the results about the hard score estimated using out-of-bang-boostrap samples in analogy to the previous objects `smooth` and `hard`.

`best_smooth` Clustering produced by the *best* method, according the specified `rankby` criterion, applied to the smooth score estimated using the empirical bootstrap sampling.

`best_hard` clustering produced by the *best* method, according the specified `rankby` criterion, applied to the hard score estimated using the empirical bootstrap sampling.

`best_obb_smooth` clustering produced by the *best* method, according the specified `rankby` criterion, applied to the smooth score estimated using the out-of-bag-boostrap samples.

`best_obb_hard` clustering produced by the *best* method, according the specified `rankby` criterion, applied to the hard score estimated using the out-of-bag-boostrap samples.

`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 that allows tracing the bootstrap sampling in almost every stage. Let `n`=sample size, `B`=bootstrap samples, `M`= number of methods in `methoset`. It contains the following objects.

`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 reports both hard and smooth scores estimated in each bootstrap replicate. `score[,1,]` reports a `code=1` if the corresponding bootstrap sample has been excluded because of errors (otherwise `code=0`).

`oob_scores`: returned if `obb=TRUE` or `obb="only"` and `savescores=TRUE`. It is an array is organized as the previous object `score` but contains information about out-of-bag-bootstrap 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)
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 Boostrap*

Description

Ranks the scores of clusters methods estimated via boostrap

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 solution are ranked. Possible values are {"lq", "mean", "1se"}. With ="lq" (default), the solutions are ranked by maximizing the estimated lower limit of the of the 1-alpha bootstrap confidence intervalfor 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 boostrap distribution.
boot_na_share	a numeric value in (0,1). During the boostrapping a method's score is set to NA if the underlying comptutation runs into errors. Methods resulting in more than B * boot_na_share errors are excluded from the comparison

Value

An S3 object of class bqs. Output components are those of bqs. See *Value* in bqs.

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

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 influence the `bqs_sol` ranking. 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.

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, and each named entry contains the corresponding clustering method in `bqs_sol$methodlist` fit on `bqs_sol$data`.

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 = 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.

Value

A list containing cluster parameters. Let P=*number of variable/features* and K=*number of clusters*. The elements of the list are as follows:

- prop: 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-8,
   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=nrows(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 for certain specifications of init (see below) where the number of cluster is retrieved from the initial partition.
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, 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 kmedian initialization.
init.tol	tolerance for the convergence of each ran of the kmedian 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 convariance 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 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" one replaces the K-medians with the K-means. With init="pam" initial clusters are determined using the PAM algorithm based on Euclidian 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 of dimension (N x {K}) containing the initial posterior probabilities that the *i*th observation belongs to the *k*th cluster. 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 (each row of W must sum up to 1). W can be seen as the initial version of the object tau described in the *Value* section below. The last alternative is to set init = f(data). Here f(data) is a function that takes data as an input and

returns the matrix with an initial hard/smooth assignment as the W matrix previously described (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 *Referenceses* 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	information on convergence and errors (see notes).
iter	number of iterations performed in the underlying EM-algorithm.
N	number of data points.
P	data dimension.
K	number of clusters.
eloglik	sample expected log-likelihood.
size	cluster size (counts).
cluster	cluster assignment based on the *maximum a posteriori* rule (MAP).
taus	a matrix of dimension (N x {K}) where tau[i, k] is the estimated posterior probability that the *i*th observation belongs to the *k*th cluster.
params	a list containing mixture components parameters. The elements of the list are as follows: $prop=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.
info	a list with two components named giving information about underlysing EM algorithm. The code obejects 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 occured. • code=-2: unexpected LAPACK routines errors occured. The flag obejects can take the following values: • flag=0 no flag. • flag=1 numerically degenerate posterior probabilities. (taus) could not be prevented. • flag=2 the ERC was enforced at least once. • flag=3 if condition of flag=1 and flag=2 occurred.

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
# In real applications set 'init.nstart' as large as possibile
set.seed(101)
fit1 <- gmix(dat, K = nc, init.nstart = 1)
print(fit1)

# 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")



# 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)
plot(x=fit2, data = dat)



# user-defined smooth "toy" initialization:
# 50% of the points are assigned to cluster 1 with probability 0.95 and to
# cluster 2 with probability 5%. The remaining data points are assigned to
# cluster 1 with probability 10% and  to cluster 2 with probability 10%
#
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)
# fit
fit3  <- gmix(dat, K = nc, init = i3)
plot(x=fit3, data = dat)



# 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)
plot(fit4, data = dat)
```

`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 object of class `qcmethod` obtained from `mset_user` and related functions

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 the `fname`. If `only_params==TRUE` (default) `fn` will return only cluster parameters (proportions, 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 the PAM setting with K=3
m <- M[[4]]
m

# cluster data with M[[3]]
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-8,
  init.nstart = 25,
  init.iter.max = 30,
  init.tol = tol)
```

Arguments

K	a vector/list, specifies the number of clusters.
init	a vector, contains the settings of the init parameter of gmix.
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.start 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.

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 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 gmix. If only_params==TRUE (default) fn will return only cluster parameters (proportions, 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 M[[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=c(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 ofkmeans.
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 more detail for a detailed description of the role of each argument and their data types.

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. If only_params==TRUE (default) fn will return only cluster parameters (proportions, 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

```
# '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 'ma1'
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`	list, contains the settings of the `medoids` parameter of pam.
`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 more detail for a detailed description of the role of each argument and their data types.

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 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 pam. If only_params==TRUE (default) fn will return only cluster parameters (proportions, 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`	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).

Requirements for `fname`. `fname` is a 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:

- `prop`: 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 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`.

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 the `fname`. If `only_params==TRUE` (default) `fn` will return only cluster parameters (proportions, 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 M[[3]]
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 M[[3]]
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(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` is expected to have the component `rankby` set.
`score`	Character vector specifying the score(s) to be plotted. Valid scores are `"hard"`, `"smooth"`, `"oob_hard"`, and `"oob_smooth"`. If `NULL` (default), all valid scores present 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.
`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).
`...`	Further arguments passed to or from other methods.

Value

A plot displaying the Bootstrap Quality Scores.

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. 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 `"boundary"`. 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 the 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.

When data is NULL (the default), the function plots only contour sets (and optionally clustering boundaries) for the estimated mixture density components, using the params information from the mbcfit object. When data is not NULL, the function additionally plots data points and their hard clustering labels, which are obtained using mbcfit to predict the cluster labels (see predict.mbcfit).

Value

A plot displaying the data with clustering information, contours, and/or boundaries, depending on the specified what argument.

See Also

gmix, plot_clustering, link{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 subset and focus the plot on specific features. 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.
what	Character vector specifying which elements to plot. Options are "clustering", "contour", and "boundary". Default is to plot "clustering" whenever cluster is not NULL, and "contour" and "boundary" whenever params is not NULL.
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.

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 `x`. 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:

`proportions` 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(data)` must be equal to `P`) or otherwise the function terminates with an error message.

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 length `nrow(data)` containing the estimated cluster labels for each observation in the provided `data`.

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 decreasing order, for any available scoring method (this is determined by the `oob` argument 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.

`mean` Method's mean (bootstrap) quadratic score.

`sterr` Method's standard error for the (bootstrap) quadratic score.

`lower_qnt` (Only shown for "mean" and "lq" ranking) 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) 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) Method's mean (bootstrap) quadratic score minus 1 standard error.

-1se (Only shown for "1se" ranking) Method's mean (bootstrap) quadratic score plus 1 standard error.

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 function handles different `code` values from the object's `info` field, each representing a specific status or error condition:

`-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 simple 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)
```

---

qscore *Clustering Quadratic Score*

Description

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

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=nrows(data) and P=ncol(data). Categorical variables and NA values are not allowed.
params	a list containing cluster parameters *(size, mean, cov)*. Let K=*number of clusters*. The elements of the list are as follows: $prop=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.
type	the type of score, a character in the set c("both", "smooth", "hard"). The default value is set to "booth". See *Details*.

Details

The function calculates quadratic scores as defined in equation (22) in Coraggio and Coretto (2023).

Value

A numeric vector with both the hard and the smooth score, or only one of them depending on the argument type.

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

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