Package: qcluster (via r-universe)

October 19, 2026

Version 3.0.0

Date 2026-09-02

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 https://lcorag.r-universe.dev

Date/Publication 2026-09-02 10:30:02 UTC

RemoteUrl https://github.com/cran/qcluster

RemoteRef HEAD

RemoteSha 8898898f5c481a64269cf844cc655a1c232f7962

Contents

apply_method . . . 2
banknote . . . 4
clust2params . . . 4
gmix . . . 5
mbind . . . 9
mset_gmix . . . 10
mset_kmeans . . . 13
mset_pam . . . 15
mset_screen . . . 17
mset_tmix . . . 20
mset_user . . . 22
plot.gmix . . . 26
plot.qcluster . . . 28
plot_clustering . . . 30
predict.gmix . . . 32
print.apply_method . . . 34
print.mset_screen . . . 34
print.qcfit . . . 35
print.qcluster . . . 36
print.qcmethod . . . 38
qcluster . . . 38
qcluster_rank . . . 43
qcluster_select . . . 45
qscore . . . 47
summary.qcmethod . . . 49
tmix . . . 50

Index **54**

`apply_method` *Fit a Single Method From a Method Set*

Description

Fit one method, selected from a method set by index or by name, on a data set. It is the refit step of `qcluster_select`, usable on its own.

Usage

```
apply_method(data, mlist, id = NULL, method_name = NULL)
```

Arguments

data	a numeric vector, matrix, or data frame of observations. Rows are observations and columns are variables/features. Categorical variables and NA values are not allowed.
mlist	a qcmethod object (or a list of such elements), e.g. the output of the mset_*() helpers or mbind. Each element exposes a fullname and a fitting function fn.
id	an integer index into mlist selecting the method to fit.
method_name	a character string matched against the method codename (the names of mlist, as shown by print).

Details

Exactly one of id or method_name must identify a single method. If both are supplied they must be consistent. The data are validated before fitting; a fitting error stops the function.

Value

An object of class apply_method. It is the native fit returned by the selected method's fitting function, with the rendered method summary prepended as the first element info_apply_method (a character vector). When the native fit is a list its components follow, flat, under their own names; otherwise it is stored under fit. The object carries the attributes method_name (the method codename) and data_name (the deparsed data argument), and its class is c("apply_method", <native class>) so native plot/predict methods still dispatch by inheritance. It prints uniformly via print.apply_method.

See Also

print.apply_method, qcluster_select, mbind, mset_user

Examples

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

# set up a couple of methods
mlist <- mbind(mset_kmeans(K = 3), mset_gmix(K = 3, erc = 100))

# fit by index (prints uniformly via print.apply_method)
fit1 <- apply_method(dat, mlist, id = 1)
fit1

# fit by codename (the names shown by print(mlist))
nm <- names(mlist)[2]
fit2 <- apply_method(dat, mlist, method_name = nm)
```

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.

clust2params *Converts Hard Assignment Into Cluster Parameters*

Description

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

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 scatter 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 (2025). The scatter 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 scatter matrix eigenvalues. It defines the so-called *eigenratio constraint*. erc=1 enforces spherical clusters with equal scatter matrices. A large erc allows for large between-cluster scatter discrepancies. The constraint bounds the relative size of the clusters and the elongation of each: it restricts the scale of a scatter against the others and the spread of a single scatter's own eigenvalues, and does not fix the shape up to scale. It is suggested to never set erc arbitrarily large; its main role is to prevent degenerate scatter 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 c("gmix", "qcfit"), with print, plot and predict methods. 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 the component 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 (2025). Consistency for constrained maximum likelihood estimation and clustering based on mixtures of elliptically-symmetric distributions. *Journal of Multivariate Analysis*.

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, subset = c(1, 2, 3),
     pch_cl = c("A", "B"), col_cl = c("#4285F4", "#0F9D58"))

## 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` `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,
  method_name = NULL
)
```

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.
method_name	optional custom codenames for the generated methods; see mset_user for the accepted forms (function, template string, or character vector). The codename is the element name (names(method_set)) used for matching and shown by print, distinct from the per-element fullname. NULL (default) keeps the automatic naming.

Details

The function produces functions implementing competing clustering methods based on several Gaussian Mixture models specifications. 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

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,
  method_name = NULL
)
```

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`.
`method_name`	optional custom codenames for the generated methods; see `mset_user` for the accepted forms (function, template string, or character vector). The codename is the element name (`names(method_set)`) used for matching and shown by `print`, distinct from the per-element `fullname`. `NULL` (default) keeps the automatic naming.

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 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

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 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,
  method_name = NULL
)
```

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`.
`method_name`	optional custom codenames for the generated methods; see `mset_user` for the accepted forms (function, template string, or character vector). The codename is the element name (`names(method_set)`) used for matching and shown by `print`, distinct from the per-element `fullname`. `NULL` (default) keeps the automatic naming.

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 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

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_screen` *Two-Step Method-Set Preselection Filter*

Description

Reduces a method set before an intensive `qcluster` run, in two steps, keeping competitive candidates with high probability (a recall-first screen). Step 1 (feasibility) is a deterministic, full-data feasibility check; Step 2 (pre-selection) is a light paired held-out screen against the reference candidate.

Usage

```
mset_screen(
  mset,
  data,
  B = 50,
  sprop = 0.5,
  score = NULL,
  delta = 2,
  max_na_prop = 0.05,
  min_cluster = NULL,
  rcond_tol = 1e-08,
  seed = NULL,
  verbose = FALSE,
  ncores = detectCores() - 2
)
```

Arguments

`mset`	a `qcmethod` object (or a list of such elements, a bare function, or a list of bare functions; anything accepted by `mbind`), the method set to screen.
`data`	a numeric vector, matrix, or data frame of observations. Rows are observations and columns are variables/features. Categorical variables and `NA` values are not allowed.
`B`	an integer `>=1`, the number of paired splits used by Step 2.

sprop	a number in (0,1), the training fraction used by Step 2, each training block having size floor(sprop * n) (see qcluster).
score	NULL or a character string in c("hard","smooth"), the driving held-out score for Step 2. NULL (default) uses the qcluster default ("smooth").
delta	a single positive number, the Step 2 admissibility-band semiwidth in sterr units (qcluster's rankby = "se" band, mean >= mean_best - delta * sterr_best). Smaller values widen the band (higher recall, less reduction); larger values prune harder. Default 2, wider than qcluster's default of 1, so that the screen keeps every candidate the final selection would admit on the same data, seed, B and sprop: under rankby = "se" this holds whenever delta here is at least the delta used there; under "mean" for any positive delta; under "lpb" when delta is at least about abs(qnorm(prob)), i.e. 1.645 at prob = 0.05 and 2.33 at prob = 0.01.
max_na_prop	a number in [0,1], the maximum tolerated split-failure fraction for a Step 2 candidate (see qcluster).
min_cluster	NULL or a positive integer, the minimum number of points a cluster must have for Step 1 to accept the fit. NULL (default) uses d + 1, with d the number of data columns.
rcond_tol	a small positive number, the reciprocal-condition-number threshold below which a cluster's scatter is treated as singular by Step 1.
seed	NULL or a single integer, a seed set before Step 2 draws its splits.
verbose	logical; if TRUE, prints a short progress message before each step.
ncores	an integer, the number of parallel workers used by Step 2's internal qcluster call; the default and the ncores = 0 "use all detected cores" convention follow qcluster. Step 2 is reproducible under seed for any ncores. Step 1 ignores this argument.

Details

Step 1 fits each candidate once on the full data and drops it on a fit error, an empty or undersized cluster, singular/near-singular scatter, or a non-finite held-out score; a warning raised during fitting flags (rather than drops) a candidate that otherwise clears every check. Step 2 runs a single qcluster call on the Step 1 survivors, so that every candidate is scored on the same splits, at rankby = "se", and keeps candidate j when its held-out mean is within delta reported dispersions of the reference candidate's mean, and its split-failure fraction does not exceed max_na_prop.

delta does not correct for the multiplicity of comparing every candidate against the best: the screen's recall degrades as the method set grows, and delta should be raised on long candidate lists. See delta for the condition under which the screen keeps everything a subsequent qcluster selection would admit.

When sterr at the reference candidate is non-finite or zero, distance is NA for every candidate, while band is still computed with the band collapsed to zero width, so that only candidates tied with the reference are admitted. band is the decision; distance is informational.

No redundancy/equivalence pruning is performed (only feasibility and relative performance).

Value

An object of class `c("mset_screen", "qcmethod")`: the retained method set, a strict subset of `mset` directly usable by `qcluster` and other downstream functions. It carries the attribute `screen_report`, a data frame with one row per input method and the Step 1 and Step 2 columns aligned on `method`:

`method` the method codename.

`stage` Step 1 outcome stage, `"fit"` or `"score"`.

`status` Step 1 status, one of `"pass"`, `"flag"`, `"drop"`.

`reason` Step 1 drop/flag reason, or `""`.

`mean` Step 2 held-out criterion (`NA` if dropped at Step 1).

`sterr` Step 2 rate-rescaled standard error of the held-out criterion, the candidate's own `qcluster()` `sterr`.

`distance` Step 2 standardized distance to the reference candidate, `(mean_best - mean) / sterr_best`, where the reference is the held-out maximizer among the candidates `qcluster` retains by `max_na_prop`; `0` at the reference itself, and `NA` for every candidate when `sterr` at the reference is non-finite or zero (see *Details*).

`band` Step 2 admissibility-band decision, the `rankby = "se"` band of `qcluster`.

`na_prop` Step 2 split-failure fraction.

`failure_pass` Step 2 failure-rate decision.

`retained` overall outcome: `TRUE` only if the method survives both steps.

Prints via `print.mset_screen`.

See Also

`print.mset_screen`, `qcluster`, `mset_user`, `mbind`

Examples

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

# a method set mixing a reasonable setup with an overfit one
mlist <- mbind(mset_kmeans(K = 2), mset_kmeans(K = 15))

# screen before an intensive qcluster() run (small B for a quick illustration)
screened <- mset_screen(mlist, dat, B = 20, seed = 1, ncores = 1)
screened
```

`mset_tmix` *Generates Methods Settings for Student-t 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 `tmix`-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_tmix(
  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,
  df = 4,
  df_fixed = TRUE,
  df_range = c(2.1, 100),
  df_warmup = 3L,
  method_name = NULL
)
```

Arguments

`K`	a vector/list, specifies the number of clusters.
`init`	settings of the `init` parameter of `tmix`. 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 `tmix` for the meaning of each initialization type.
`erc`	a vector/list, contains the settings of the `erc` parameter of `tmix`.
`iter_max`	a integer vector, contains the settings of the `iter_max` parameter of `tmix`.
`tol`	a vector/list, contains the settings of the `tol` parameter of `tmix`.
`init_nstart`	a integer vector, contains the settings of the `init_nstart` parameter of `tmix`.
`init_iter_max`	a integer vector, contains the settings of the `init_iter_max` parameter of `tmix`.
`init_tol`	a vector/list, contains the settings of the `init_tol` parameter of `tmix`.
`df`	a numeric vector, contains the settings of the `df` parameter of `tmix`. Each value is recycled to all `K` components within a given configuration. With `df <= 2` the component covariance does not exist and the scores of that configuration are `NA` (see *Details*).

df_fixed	a logical vector, contains the settings of the df_fixed parameter of tmix.
df_range	numeric length-2 vector c(lower, upper), the df_range parameter of tmix. Held fixed across all generated configurations (not expanded like the other arguments).
df_warmup	a integer vector, contains the settings of the df_warmup parameter of tmix.
method_name	optional custom codenames for the generated methods; see mset_user for the accepted forms (function, template string, or character vector). The codename is the element name (names(method_set)) used for matching and shown by print, distinct from the per-element fullname. NULL (default) keeps the automatic naming.

Details

The function produces functions implementing competing clustering methods based on several Student-t Mixture models specifications. This is a specialized version of the more general function mset_user. In particular, it produces a list of tmix functions each corresponding to a specific setup in terms of both model hyper-parameters (*e.g.* the number of clusters, the eigenvalue ratio constraint, the degrees of freedom, *etc.*) and algorithm's control parameters (*e.g.* the type of initialization, maximum number of iteration, *etc.*). See tmix 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.

The matrix scored for a tmix configuration is the covariance, df/(df-2) times the scatter, which exists only for df > 2 (see tmix). A configuration with a component at df <= 2 is generated all the same; its scores in qscore and qcluster are NA.

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 tmix 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, save_smd, 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 tmix. If only_params==TRUE, fn will return only cluster parameters proportion, mean, and cov (the covariance, not the scatter matrix).

References

Coretto, Pietro and Christian Hennig (2025). Consistency for constrained maximum likelihood estimation and clustering based on mixtures of elliptically-symmetric distributions. *Journal of Multivariate Analysis*.

See Also

tmix, mset_user

Examples

```
# 'tmix' settings combining number of clusters K={2,3} and degrees of
# freedom df={5,10}
A <- mset_tmix(K = c(2,3), df = c(5,10))

# select setup 1: K=2, df=5, 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
```

---

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 a set of tuned methods and algorithms. The *base* clustering methodology is provided via a user-defined function. The latter prototype is expanded in a list of functions 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, ..., method_name = 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*).

`method_name` optional custom codenames for the generated methods. The codename is the element name, i.e. `names(method_set)[i]`, used for matching and shown by `print`; it is distinct from the per-element `fullname` descriptor shown by `summary.qcmethod`. Accepts three forms: a **function** invoked once per grid point with that point's tuning arguments (only declared formals are passed when the function has no `...`); a **template string** containing `{argname}` placeholders replaced by argument values; or a **character vector** of length equal to the number of generated methods (length one is recycled). When `NULL` (the default), the current automatic naming (`fname_i`) is used. Duplicates are disambiguated by appending the element position within each colliding group.

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 `qcluster` 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 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 use:
#   of the optional arguments '...' passed to the underlying 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"))

# custom codenames via a function (only uses the arguments it declares)
A2 <- mset_user(fname="hc_wrapper", K = c(2,3), method = c("ward.D", "complete"),
                method_name = function(method, K) paste0("hc_", method, "_K", K))

# 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 underlying
#    '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.gmix` *Plot Fitted Mixture Models*

Description

Plot methods for the fits returned by `gmix` and `tmix`: a wrapper around `plot_clustering` drawing clustering assignments, contours and boundaries.

Usage

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

```
  pch_cl = NULL,
  ...
)

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

Arguments

x	An object of class qcfit, typically a result of the gmix or tmix 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". Features that cannot be drawn with the available inputs are dropped with a warning (see plot_clustering).
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

A plot is produced only for a successful fit, i.e. when x$info$code is 1 or 2. The features that can be drawn depend on the components stored in x, in particular x$params and x$cluster.

With data, the function passes it together with the stored parameters and the labels in x$cluster to plot_clustering. Without data, a synthetic frame spanning the components is built from x$params, and the contour level of each component is the median of its score under the model, taken at the chi-square median of the squared Mahalanobis distance.

For a tmix fit the "boundary" layer shades the Student-t rule of the model, the one predict uses.

Value

Called for its side effects.

See Also

gmix, tmix, plot_clustering, predict.gmix

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.qcluster *Plot Held-Out Quadratic Score Results*

Description

Produce a plot of the held-out quadratic scores summarized in a qcluster object.

Usage

```
## S3 method for class 'qcluster'
plot(
  x,
  score = NULL,
  perc_scale = FALSE,
  nmax = NULL,
  magnify = NULL,
  annotate = NULL,
  ...
)
```

Arguments

x	An S3 object of class qcluster as returned by the qcluster 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" and "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.
nmax	Numeric; maximum number of methods to display, keeping only the top-ranked ones. Must be a single positive integer no larger than the total number of methods. If NULL (default), all methods are shown.
magnify	Numeric; specifies the number of top-ranked methods whose bars are drawn with magnified height for visual emphasis. Must be a single number less than or equal to the number of displayed methods. If NULL (default), the number is determined automatically from the score values. Methods are emphasized visually, not filtered; use nmax to limit the displayed set.
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

qcluster

Examples

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

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

# run the held-out validation
# change B and ncores to a much larger value in real problems
res <- qcluster(dat, mlist, sprop = 0.632, B = 10, type = "both", rankby = "se",
           ncores = 1, save_scores = FALSE)

# Plot with default settings
```

```
plot(res)

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

`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, one per row of `data`. Required by the `"clustering"` layer; it also defines the clusters whose contours are drawn (see *The contour layer*).
`params`	A list with `proportion`, `mean` and `cov`, as returned by `clust2params` or stored in a fitted model. Required by the `"contour"` and `"boundary"` layers, which need 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.

The boundary layer

The `"boundary"` layer shades the decision regions of the quadratic discriminant score: a grid point is assigned to the component maximizing

$$qs_k(x) = \log \pi_k - \tfrac{1}{2} \log \det \Sigma_k - \tfrac{1}{2}(x - \mu_k)' \Sigma_k^{-1} (x - \mu_k),$$

the quantity the hard score of Coraggio and Coretto (2023) maximizes when it assigns a point. `plot` on a `tmix` fit shades the Student-t rule of the model instead, the one `predict` uses.

The contour layer

The `"contour"` layer draws one level set of qs_k for each cluster: an ellipse centred at μ_k with the shape of Σ_k, at the level c_k equal to the median of qs_k over the points of the cluster, written on the ellipse. The ellipse encloses half of the cluster's points; a heavier or more concentrated component scores its points higher, and the label shows it. The clusters are those given in `cluster` or, without it, those of the boundary rule. In a scatterplot matrix the level of each panel is the median of the score of the two variables shown. In one dimension the layer draws the weighted component densities.

See Also

`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.gmix` *Predict Cluster Assignments From a Fitted Mixture Model*

Description

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

Usage

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

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

Arguments

`object`	An object of class `qcfit`, representing the fitted mixture model. This is typically obtained in output from the `gmix` or `tmix` 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 methods use the parameters of a fitted `qcfit` model to allocate new data points to the estimated clusters, after checking that the fit holds valid estimates and that `newdata` has the model's dimension `P`.

For **Gaussian** models (`gmix`), the `qcfit` 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 the cluster scatter matrices, which for a Gaussian component are its covariances.

For **Student-t** models (tmix), the params list must additionally contain:

scatter A numeric array of dimensions c(P, P, K), representing the scatter matrices, i.e. the scale matrices appearing in the Student-t density, which the MAP rule uses instead of cov.

and the object must carry a df component (length-K vector of degrees of freedom).

For Gaussian models, the stored mixture parameters must be Gaussian-ready, i.e. proportions must be positive and sum to 1, and the scatter matrices must be finite, symmetric, and positive definite.

The predicted clustering is obtained as the MAP estimator using posterior weights of the fitted mixture model. For Gaussian mixtures, denoting with $z(x)$ the predicted cluster label for point x, and with ϕ the (multivariate) Gaussian density:

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

For Student-t mixtures, the multivariate Student-t density replaces the Gaussian density in the MAP rule, using the scatter matrices and degrees of freedom.

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, tmix

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.apply_method` *Print a Fitted Method (apply_method Result)*

Description

Uniform console display for the object returned by `apply_method`, independent of the underlying clustering routine: the method codename and the data set it was fitted on, the method settings, and the components available in the object.

Usage

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

Arguments

`x`	An object of class `apply_method`.
`...`	Additional arguments passed to or from other methods.

Value

`x`, invisibly.

See Also

`apply_method`, `qcluster_select`

`print.mset_screen` *Display a Summary of a Method-Set Screening Result*

Description

Print method for objects of class `mset_screen`, produced by `mset_screen`. It prints the Step 1/Step 2 counts and the retained methods.

Usage

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

Arguments

x	An object of class mset_screen.
...	Additional arguments passed to or from other methods.

Value

x, invisibly.

See Also

mset_screen, print.qcmethod

Examples

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

mlist <- mbind(mset_kmeans(K = 2), mset_kmeans(K = 15))
screened <- mset_screen(mlist, dat, B = 20, seed = 1, ncores = 1)
print(screened)
```

---

print.qcfit *Display Information for Mixture Model Objects*

Description

This function provides a print method for objects of class qcfit, returned in output by the gmix or tmix functions.

Usage

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

Arguments

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

Details

The print.qcfit function gives a summary of a model-based clustering fit, estimated using the gmix or tmix function.

The printed message depends on x$info$code:

-2 'Lapack DSYEV failed'. This error occurs whenever any of the cluster scatter matrices (the covariances, for gmix) becomes singular during estimation.

`-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` The algorithm 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 `qcfit` 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`, `tmix`

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)
```

---

print.qcluster *Display Information on Held-Out Quadratic Score Objects*

Description

Print method for objects of class `qcluster`, produced by `qcluster` or `qcluster_rank`. It prints a summary of the held-out quadratic score results for the clustering solutions considered.

Usage

```
## S3 method for class 'qcluster'
print(x, max_raw = 10, ...)
```

Arguments

x	An object of class qcluster, the output of qcluster or of qcluster_rank.
max_raw	Controls how many methods are listed per score table, in increasing rank order. A positive integer prints the top max_raw by rank (default 10); Inf prints all methods; the string "selected" prints, for an object ranked by "se" (i.e. carrying a selected column), all methods with selected == TRUE (with a warning and a fallback to the default otherwise). Any other value is rejected.
...	Additional arguments passed to or from other methods.

Details

The printed summary tables have row.names set to the method codename and show, along the columns: id (method index in method_set), rank, mean (held-out criterion), sterr, lower and upper (the percentile bounds), complexity, crank (complexity rank), selected (only for the "se" ranking), and na_prop. Methods with missing ranks are omitted; if all ranks are missing for a score component, a short message is printed instead of a table.

Value

No return value, called for side effects.

See Also

qcluster, qcluster_rank

Examples

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

KM    <- mset_kmeans(K = 2:5)
GMIX  <- mset_gmix(K = 2:5, erc = c(1, 50, 100))
mlist <- mbind(KM, GMIX)

res <- qcluster(dat, mlist, sprop = 0.632, B = 100, type = "both", rankby = "se",
           ncores = 1, save_scores = TRUE)

# default: top 10 by rank
res

# show all methods
print(res, max_raw = Inf)

# show only the methods in the se selection band
print(res, max_raw = "selected")
```

print.qcmethod *Display a Compact Summary of a Method Set*

Description

Prints a compact table for objects of class qcmethod, showing the method id, the codename (method_name), the underlying base routine (fname), and the tuning arguments that vary within each routine group.

Usage

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

Arguments

x	An object of class qcmethod.
...	Additional arguments passed to or from other methods.

Value

x, invisibly.

See Also

mset_user, mset_kmeans, mset_gmix, mset_pam, summary.qcmethod

Examples

```
data("banknote")

KM <- mset_kmeans(K = 2:4)
print(KM)
```

qcluster *Held-Out Validation of Clustering Solutions by Quadratic Scoring*

Description

Estimates the held-out quadratic criterion and the complexity gap of clustering solutions provided by a list of candidate models, methods or algorithmic settings. The procedure is held-out validation: training blocks of a fixed size are drawn repeatedly without replacement, and each fitted solution is scored on the complementary validation block. The held-out criterion is the estimator; the estimand it targets is the generalization score, the expected population score of a solution fitted at that training size.

Usage

```
qcluster(
  data,
  method_set,
  sprop = 0.5,
  B = 100,
  type = "smooth",
  ncores = detectCores() - 2,
  prob = 0.05,
  rankby = "mean",
  delta = 1,
  max_na_prop = 0.05,
  save_scores = TRUE
)
```

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.
method_set	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.
sprop	a number in (0,1), the training fraction. Each training block has size n_sub = floor(sprop * n) and is drawn without replacement; the complementary validation block carries the remaining n - n_sub observations.
B	an integer >=1, the number of splits (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.
ncores	an integer, it defines the number of cores used for parallel computing (see *Details*). Setting ncores = 0 uses all detected cores.
prob	a number in (0,1), the one-sided tail probability locating the percentile bounds reported for the held-out criterion: lower is the prob quantile and upper the 1 - prob quantile of the rate-rescaled centred deviations. Default 0.05.
rankby	character string in c("mean", "lpb", "se"). Ranking is mandatory. "mean" (default) maximizes the held-out criterion, "lpb" maximizes the lower percentile bound, and "se" applies the dispersion-band parsimonious rule (see delta). See qcluster_rank, and *Details* for how each rule ranks and breaks ties.
delta	a single positive number, the band semiwidth in sterr units used by rankby = "se": a method is admissible when mean >= mean_best - delta * sterr_best, where best is the held-out maximizer among the methods retained by max_na_prop and sterr_best is that method's own rate-rescaled standard error. Default 1. delta is read in units of sterr, that is sqrt(n_sub / n) times the across-split standard deviation of the held-out score, not divided by sqrt(B): delta = 1 is

not the one-standard-error rule of cross-validation. Ignored outside `rankby = "se"`.

`max_na_prop` a numeric value in `[0,1]`. During the validation a method's score is set to `NA` if the underlying computation runs into errors. Methods whose proportion of failed splits exceeds `max_na_prop` are excluded from ranking and selection. The columns of a surviving method are computed over its successful splits only, so two methods need not rest on the same splits; read `na_prop` alongside the other columns.

`save_scores` logical, if `=TRUE` (default) the raw per-split scores are retained in `out$raw`.

Details

The function implements the estimation and selection of an appropriate clustering based on the quadratic-scoring methodology proposed in Coraggio and Coretto (2023), here by held-out validation. For each of `B` splits a candidate is fitted on a training block `D_b` of size `n_sub = floor(sprop * n)`, drawn without replacement, and scored both on `D_b` (apparent score) and on the validation block `Y_b = data \ D_b` (held-out score). The per-method criterion is the mean held-out score; the apparent-held-out gap is the complexity gap, reported as `complexity`. The gap is descriptive: comparable across the candidates of one run, not necessarily non-negative, and used to separate candidates that agree on the held-out criterion.

The uncertainty of the held-out criterion is the size-`n` sampling standard error, estimated by the rate transfer `sqrt(n_sub / n)` times the across-split standard deviation (*not* divided by `sqrt(B)`), and a pair of Politis-Romano subsampling percentile bounds whose centred offsets are rescaled by the same rate factor and placed at the `prob` and `1 - prob` quantiles.

The held-out mean is, in expectation, exactly the generalization score at training size `n_sub`, provided every split is scored. A failed split is discarded, and whether a fit fails depends on the training block drawn, so the mean of a method with a high `na_prop` rests on a subset of splits selected by its own failures.

Scope of the bounds. The columns `sterr`, `lower` and `upper` are returned for every engine, but the inferential reading rests on a limit theory established only for some of them: constrained mixture maximum likelihood, k-means and trimmed k-means are covered. PAM, boundary mixture solutions, and arbitrary user methods supplied through `mset_user` are not, and for those the three columns are descriptive summaries of the across-split spread.

Retention, ranking and ties. A method enters the ranking only if it is retained, that is if `na_prop <= max_na_prop` *and* its `mean` is finite; a non-retained method keeps `NA` in `rank` and `crank`. Under `rankby = "se"` the band is built around the held-out maximizer among the retained methods, and a non-finite `sterr` at that maximizer collapses the band to zero width, so that only ties at the maximizer are admitted. Under `"mean"` and `"lpb"` the ordering applies a three-key cascade: the criterion descending, then `sterr` ascending, then `K` ascending, the last key read from the method's own `callargs$K`; a method without `K` in its call arguments (typically a `mset_user` prototype) loses every tie-break on that key. Methods tied across all three keys collapse to the minimum (competition) rank, which leaves gaps in the rank sequence. Under `"se"`, by contrast, the ranks are distinct and consecutive, with no collapsing of ties.

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. 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 method_set list and repeat the estimation with a value of B that is as large as possible relative to available computational resources.

The default is B=100. The percentile bounds (lower and upper, and with them the rankby="lpb" ranking) require a sufficiently large B: with small B the extreme centred quantiles are noisy and these become unstable. The standard-error rule rankby="se" is comparatively robust at small B.

Parallel computing. The splits are processed using foreach-based parallel computation via the doParallel parallel backend. Note that depending on the system settings, and how the functions in method_set 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, qcluster avoids this oversubscription by forcing each worker to use a single BLAS/OpenMP thread whenever process-level parallelism is active. Each (method, split) task draws its own L'Ecuyer-CMRG RNG substream from the caller's RNG state, so a call is reproducible under set.seed() for any ncores, also for methods with internal randomness (e.g. the nstart restarts of kmeans(), the initialization of gmix()).

method_set **argument.** The method_set 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.

Ranking is always performed. The best_* components are lightweight descriptors of the rank-1 method (id, method_name, crank); the corresponding model is fitted on demand via qcluster_select or apply_method.

Value

An S3 object of class qcluster. The exact components depend on type, rankby, and save_scores.

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

id index of the method in method_set;

rank rank according to rankby;

mean held-out criterion, the mean of the held-out scores;

sterr rate-rescaled standard error of the held-out criterion;

lower the prob percentile bound for the held-out criterion;

upper the 1 - prob percentile bound for the held-out criterion;

complexity complexity gap: the average over splits of the apparent score minus the held-out score, how much a candidate flatters itself when scored on the block it was fitted on;

`crank` complexity rank (`1` = least complex), over the methods retained by `max_na_prop`;

`na_prop` proportion of failed splits, in `[0,1]`. A split counts as failed when *either* the apparent or the held-out score is missing, and every column above rests on the same set of splits;

Other output components are:

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

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

best_smooth list with components `id` (index of the method in `method_set`), `method_name` (the method codename, i.e. its `method_set` name), and `crank` (the method complexity rank) for the best smooth-scoring method. Returned only when a rank-1 solution exists.

best_hard analogous object for the hard score.

selected when `rankby` is `"se"`, each score summary gains a logical column `selected` (after `crank`) flagging the methods in the admissibility band of the rule.

origin a character tag equal to `"qcluster"` for a direct `qcluster()` output.

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

B the number of splits.

sprop the training fraction.

prob the one-sided tail probability locating the reported percentile bounds.

method_set the elements of `method_set` for which a solution is produced.

rankby the ranking criterion.

max_na_prop the failed-split threshold used to gate ranking and selection.

raw a list returned if `save_scores=TRUE`. Let `n`=sample size, `n_sub`=training-block size, `B`=number of splits, `M`= number of methods in `method_set`. It contains:

`sub_id`**:** an array of dimension `n_sub` x `B` where the `j`-th column contains the indexes of the observed data points forming the `j`-th training block. That is, `data[sub_id[, j], ]` gives the `j`-th training block.

`scores`**:** an array of dimension `(M` x `5` x `B)` storing, for each method and each split, an error code together with the apparent and held-out hard/smooth scores.

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)

# run the held-out validation
# change B and ncores to a much larger value in real problems
res <- qcluster(dat, mlist, sprop = 0.632, B = 10, type = "both", rankby = "se",
           ncores = 1, save_scores = TRUE)
names(res)
res
```

---

qcluster_rank *Ranking Clustering Solutions Scored by Held-Out Validation*

Description

Re-ranks the held-out quadratic scores of a qcluster object under a different rule, without recomputing anything, and returns a display-only ranked object.

Usage

```
qcluster_rank(
  object,
  type = NULL,
  rankby = "mean",
  prob = NULL,
  max_na_prop = NULL,
  delta = NULL
)
```

Arguments

object	an object of class qcluster, a direct output of qcluster. A display-only ranked object (of class qcluster_rank) is not accepted; re-rank from the original qcluster() output instead.
type	NULL or a character vector selecting the score components to rank, a subset of c("hard", "smooth") (or "both"). NULL (default) inherits the components present in object. If a requested component is absent, the call stops.

`rankby` character string specifying how the scored solutions are ranked, one of `c("mean", "lpb", "se")`. With `="mean"` (default) the solutions are ranked by maximizing the held-out criterion. With `="lpb"` they are ranked by maximizing the lower percentile bound. With `="se"` the dispersion-band parsimonious rule selects the least-complex method within `delta` reported dispersions of the held-out maximizer. The retention gate, the three-key cascade used by the two maximizing rules, and the tie behavior of each rule are documented in `qcluster`; see *Details* here for small-`B` behavior.

`prob` `NULL` or a number in `(0,1)`, the one-sided tail probability locating the percentile bounds: `lower` is the `prob` quantile and `upper` the `1 - prob` quantile. `NULL` (default) inherits the value stored in `object`. A value differing from the stored one recomputes the `lower`/`upper` bounds from the raw per-split scores and is therefore honored only when `object` retains `raw` (i.e. was built with `save_scores = TRUE`).

`max_na_prop` `NULL` or a numeric value in `[0,1]`. `NULL` (default) inherits the value stored in `object`. Methods whose proportion of failed splits exceeds `max_na_prop` are excluded from ranking. Re-applying the gate uses the stored `na_prop` column and needs no raw scores.

`delta` `NULL` or a single positive number, the band semiwidth in `sterr` units used by `rankby = "se"` (see `qcluster`). `NULL` (default) inherits the value stored in `object`. An override needs no raw scores.

Details

`NULL` arguments inherit the corresponding value stored in `object`; a supplied value overrides it. A `prob` override requires the raw per-split scores and is therefore restricted to a `qcluster()` output built with `save_scores = TRUE`; a `delta` override needs no raw scores and works on any object (see `delta`).

The `rankby="lpb"` ranking and the reported `lower`/`upper` bounds require a sufficiently large `B`: with small `B` the extreme centred quantiles are noisy and these become unstable. The dispersion-band rule `rankby="se"` is comparatively robust at small `B`.

Value

An S3 object of class `c("qcluster_rank", "qcluster")` (a display-only ranked object). It carries the re-ranked score summaries (with recomputed `rank`, `crank`, and, under `rankby = "se"`, a logical `selected` column after `crank`), refreshed `best_*` descriptors (`list(id, method_name, crank)` with `method_name` the method codename, only when a rank-1 solution exists), the effective `rankby`/`type`/`prob`/`max_na_prop`/`delta`, and `method_set`. Unlike the direct `qcluster()` output, a ranked object does *not* carry `raw` nor the data matrix and cannot be used to refit; run selection on the original `qcluster()` output (see `qcluster_select`).

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

qcluster, qcluster_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)

# run the held-out validation
# change B and ncores to a much larger value in real problems
res <- qcluster(dat, mlist, sprop = 0.632, B = 10, rankby = "mean", ncores = 1)
res

# now change ranking criterion (produces a display-only ranked object)
res2 <- qcluster_rank(res, rankby = "se")
res2
```

---

qcluster_select *Select Ranked Clustering Solutions by Held-Out Quadratic Score*

Description

Fit clustering solutions referenced by a direct `qcluster()` object, either by rank (selection mode) or directly by method index/name (wrapper mode). The model is always (re-)estimated on the full data set.

Usage

```
qcluster_select(
  object,
  rank = 1,
  type = NULL,
  rankby = NA,
  prob = NULL,
  max_na_prop = NULL,
  delta = NULL,
  id = NULL,
  method_name = NULL,
  data = NULL
)
```

Arguments

object	a direct output of qcluster. A display-only ranked object (class qcluster_rank, produced by qcluster_rank) is not accepted; run selection on the original qcluster() output instead.
rank	an integer >=1 specifying the rank of the solution to select in selection mode. Default is 1.
type	a character string specifying the score component, "hard" or "smooth". NULL (default) uses the only component present; when object carries both, type is required.
rankby	a character string in c("mean", "lpb", "se"), or NA (default). In selection mode, NA uses the ranking already present in object; a non-NA value re-ranks on the fly without mutating object.
prob	NULL (default) or a number in (0,1), the one-sided tail probability locating the percentile bounds, used to retune them when re-ranking on the fly (requires save_scores = TRUE).
max_na_prop	NULL (default) or a numeric value in [0,1]. NULL inherits the value stored in object; a supplied value re-applies the failed-split gate.
delta	NULL (default) or a single positive number, the band semiwidth in sterr units used when re-ranking on the fly with rankby = "se" (see qcluster). NULL inherits the value stored in object; a supplied value re-applies the band and needs no raw scores.
id	an integer index into object$method_set. Supplying id or method_name triggers wrapper mode (ranking is ignored).
method_name	a character string matched against the method codename (the names of object$method_set, as shown by print). See id.
data	optional data set on which to (re-)fit. When omitted, the refit uses the matrix stored in object$data$data. When supplied it overrides the stored matrix and must have the same number of features as the scored object.

Details

Data resolution. By default the refit uses the matrix stored in object$data$data. An explicit data argument overrides it and is validated to have the same number of features (columns) as the scored object.

Selected solutions are always re-estimated on the full data set; for stochastic methods, control reproducibility with set.seed().

Value

A single selected solution is returned directly: an object of class apply_method (see apply_method), or, if its full-data refit failed, an object of class qcluster_select_error holding the failure status and message. When several tied solutions are selected they are returned as a named list of such objects (names being the method codenames) and a message announces the available components. NULL is returned when no solution matches the request. Wrapper mode always addresses a single method and so returns the fit directly. In selection mode, if the requested rank exceeds the largest available rank the worst available rank is returned instead; if it is within range but absent because of rank gaps, NULL is returned.

See Also

qcluster, qcluster_rank, apply_method

Examples

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

# set up methods
KM   <- mset_kmeans(K = 2:5)
GMIX <- mset_gmix(K = 2:5, erc = c(1, 50, 100))
mlist <- mbind(KM, GMIX)

# run the held-out validation (the full scored object)
res <- qcluster(dat, mlist, sprop = 0.632, B = 20, type = "both", rankby = "se",
           ncores = 1, save_scores = TRUE)

# selection mode: best smooth solution (data reused from the object)
best <- qcluster_select(res, rank = 1, type = "smooth")
best

# wrapper mode: fit a specific method by index (returns the fit directly)
fit <- qcluster_select(res, id = 1)

# re-rank on the fly by a different criterion
sel <- qcluster_select(res, rank = 1, type = "smooth", rankby = "mean")
```

---

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
```

---

summary.qcmethod *Inspect a Method Set or a Single Method*

Description

Without id or method_name prints the same compact table as print.qcmethod. With one of them, prints the full settings of that method (its fullname and all callargs entries).

Usage

```
## S3 method for class 'qcmethod'
summary(object, id = NULL, method_name = NULL, ...)
```

Arguments

object	An object of class qcmethod.
id	an integer index into object selecting the method to summarize.
method_name	a character string matched against the method codename (the names of object, as shown by print).
...	Additional arguments passed to or from other methods.

Details

Exactly one method must be identified; if both id and method_name are supplied they must agree. The same two arguments identify a method in apply_method and qcluster_select.

Value

object, invisibly.

See Also

print.qcmethod, apply_method, mset_user

Examples

```
data("banknote")

KM <- mset_kmeans(K = 2:4)
summary(KM)
summary(KM, id = 1)
summary(KM, method_name = "kmeans_K2")
```

`tmix` *Student-t Mixture Modelling*

Description

Fast implementation of the ECM algorithm for ML estimation and clustering of Student-t mixture models with scatter matrix regularization based on eigenvalue ratio constraints. The constraint acts on the scatter matrices, the scale matrices of the Student-t density; the covariance, `df/(df-2)` times the scatter, exists only for `df > 2`.

Usage

```
tmix(
  data,
  K = NA,
  erc = 50,
  iter_max = 1000,
  tol = 1e-08,
  init = "kmed",
  init_nstart = 25,
  init_iter_max = 30,
  init_tol = tol,
  df = 4,
  df_fixed = TRUE,
  df_range = c(2.1, 100),
  df_warmup = 3L,
  save_cluster = TRUE,
  save_params = TRUE,
  save_taus = FALSE,
  save_smd = 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 ECM algorithm.
`tol`	tolerance for the convergence of the ECM 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.
`df`	degrees of freedom for the Student-t components. A scalar (recycled to all `K` components) or a length-`K` vector. When `df_fixed = TRUE` (the default) these values are held constant; otherwise they are the starting values for estimation. See *Details*.
`df_fixed`	logical. If `TRUE` the degrees of freedom are held fixed throughout the estimation. If `FALSE` they are estimated per component using a golden-section search on the Q_2 objective (McLachlan and Peel, 2000).
`df_range`	numeric length-2 vector giving the search interval `c(lower, upper)` for the estimated degrees of freedom. The lower bound must be `> 2`.
`df_warmup`	non-negative integer. Degrees of freedom are estimated only after this many completed ECM iterations. Ignored when `df_fixed = TRUE`.
`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.
`save_smd`	logical, if `TRUE` the matrix of squared Mahalanobis distances based on the scatter matrices is returned.

Details

The function implements the constrained ML estimator for Student-t mixtures studied in Coretto and Hennig (2025). The Student-t is treated as a Gaussian scale mixture: each ECM iteration refreshes the latent gamma weights conditional on the just-updated location and scatter (ECM acceleration), enforces the eigenvalue ratio constraint via a golden-section conditional M-step, and optionally updates per-component degrees of freedom by maximizing the Q_2 objective of McLachlan and Peel (2000, Eq. 7.9).

This function uses highly optimized C code for fast execution. The constrained M-step extensively uses low-level BLAS/LAPACK routines.

Initialization. The initialization methods are the same as for `gmix`. See the documentation of `gmix` for details on the available initialization strategies.

Eigenvalue ratio constraint (`erc`)**.** The eigenvalue ratio constraint is applied to the scatter matrices, not to the covariances, and works the same as for `gmix`. See the documentation of `gmix` for details.

Scatter and covariance. The scatter is the scale matrix of the Student-t density, the matrix the estimator constrains, and is always returned. The covariance is the second central moment, `df/(df-2)` times the scatter, and exists only for `df > 2`: for a component with `df <= 2` the `$cov` block is `NA` and a warning names it. `qscore` and `qcluster` score the covariance, so such a component yields `NA` scores.

Degrees of freedom (df). Setting df_fixed = TRUE (the default) holds the degrees of freedom constant at the values specified in df. This is the setting covered by the consistency theory of Coretto and Hennig (2025). Setting df_fixed = FALSE enables per-component estimation of the degrees of freedom using a golden-section search on the Q_2 conditional M-step objective. Estimation starts after df_warmup completed iterations.

Value

An S3 object of class c("tmix", "qcfit"), with print, plot and predict methods. Output components are as follows:

info a list with two components named code and flag giving information about the underlying ECM 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 ECM 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 location parameters; $cov= array of size (P x P x K) containing the component covariances, df/(df-2) times the scatter matrices; NA, with a warning, for a component with df <= 2 (see *Details*); $scatter= array of size (P x P x K) containing the scatter matrices, the scale matrices of the Student-t density, on which the eigenvalue ratio constraint acts.

df a length-K vector of component degrees of freedom.

smd a matrix of dimension (N x K) of squared Mahalanobis distances based on the scatter matrices. Returned when save_smd = TRUE.

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 (2025). Consistency for constrained maximum likelihood estimation and clustering based on mixtures of elliptically-symmetric distributions. *Journal of Multivariate Analysis*.

McLachlan, Geoffrey J. and David Peel (2000). *Finite Mixture Models*. Wiley.

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 <- tmix(dat, K = nc, init_nstart = 1)
print(fit1)

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

# fit with estimated degrees of freedom
set.seed(102)
fit2 <- tmix(dat, K = nc, df = 5, df_fixed = FALSE, init_nstart = 1)
print(fit2)
fit2$df  # estimated degrees of freedom

## End(Not run)

# user-defined initialization with hard assignment labels
set.seed(103)
i2   <- sample(1:nc, size = n, replace = TRUE)
fit3 <- tmix(dat, K = 2, init = i2)
## Not run:
plot(x = fit3, data = dat)

## End(Not run)
```

Index

∗ **datasets**
banknote, 4

apply_method, 2, *34*, *41*, *46, 47*, *49*

banknote, 4

clust2params, 4, *10*, *12*, *14*, *16*, *23, 24*, *30, 31*, *42*, *48*

foreach, *23*

gmix, 5, *10–12*, *26–28*, *32, 33*, *35, 36*, *48*, *51*

kmeans, *13, 14*

mbind, *3*, 9, *17*, *19*, *39*, *42*
mset_gmix, 10, *10*, *23, 24*, *38*, *42*
mset_kmeans, *10*, 13, *23, 24*, *38*, *42*
mset_pam, *10*, 15, *23, 24*, *38*
mset_screen, 17, *34, 35*
mset_tmix, 20
mset_user, *3*, *9–16*, *19*, *21*, 22, *22*, *38*, *40*, *42*, *49*

pam, *15, 16*, *42*
plot, *31*
plot.gmix, 26
plot.qcluster, 28
plot.tmix *(*plot.gmix*)*, 26
plot_clustering, *26–28*, 30
predict, *27*, *31*
predict.gmix, *28*, 32
predict.tmix *(*predict.gmix*)*, 32
print.apply_method, *3*, 34
print.mset_screen, *19*, 34
print.qcfit, 35
print.qcluster, 36
print.qcmethod, *35*, 38, *49*

qcluster, *17–19*, *21*, *23*, *29*, *36, 37*, 38, *43–47*, *51*
qcluster_rank, *36, 37*, *39*, 43, *46, 47*
qcluster_select, *2, 3*, *34*, *41*, *44*, 45, *45*, *49*
qscore, *5*, *21*, 47, *51*

summary.qcmethod, *23*, *38*, 49

tmix, *20–22*, *26–28*, *31–33*, *35, 36*, 50