Package 'MEDseq'

July 21, 2025

Type Package

Date 2025-03-10

Title Mixtures of Exponential-Distance Models with Covariates

Version 1.4.2

Description Implements a model-based clustering method for categorical life-course sequences relying on mixtures of exponential-distance models introduced by Murphy et al. (2021) <doi:10.1111/rssa.12712>. A range of flexible precision parameter settings corresponding to weighted generalisations of the Hamming distance metric are considered, along with the potential inclusion of a noise component. Gating covariates can be supplied in order to relate sequences to baseline characteristics and sampling weights are also accommodated. The models are fitted using the EM algorithm and tools for visualising the results are also provided.

Depends R (>= 4.0.0)

License GPL (>= 3)

Encoding UTF-8

URL https://cran.r-project.org/package=MEDseq

BugReports https://github.com/Keefe-Murphy/MEDseq/issues

LazyData true

Language en-GB

Imports cluster, matrixStats (>= 1.0.0), nnet (>= 7.3-0), seriation, stringdist, TraMineR (>= 2.2-10), WeightedCluster

Suggests knitr, rmarkdown, viridisLite (>= 0.4.0)

RoxygenNote 7.3.2

VignetteBuilder knitr

NeedsCompilation no

Author Keefe Murphy [aut, cre] (ORCID: <https://orcid.org/0000-0002-7709-3159>), Thomas Brendan Murphy [ctb] (ORCID: <https://orcid.org/0000-0002-5668-7046>), Raffaella Piccarreta [ctb] (ORCID:

<https://orcid.org/0000-0002-8876-3656>),
Isobel Claire Gormley [ctb] (ORCID:
<https://orcid.org/0000-0001-7713-681X>)

Maintainer Keefe Murphy <keefe.murphy@mu.ie>

Repository CRAN

Date/Publication 2025-03-10 10:40:02 UTC

Contents

MEDseq-package 2
biofam 4
dbs 6
dist_freqwH 8
get_MEDseq_results 9
MEDseq_AvePP 11
MEDseq_clustnames 12
MEDseq_compare 15
MEDseq_control 18
MEDseq_entropy 23
MEDseq_fit 25
MEDseq_meantime 31
MEDseq_news 33
MEDseq_stderr 33
mvad 35
plot.MEDseq 37
predict.MEDgating 45
wKModes 47

Index **51**

`MEDseq-package` *MEDseq: Mixtures of Exponential-Distance Models with Covariates*

Description

Fits MEDseq models: mixtures of Exponential-Distance models with gating covariates and sampling weights. Typically used for clustering categorical/longitudinal life-course sequences.

Details

Type: Package

Package: **MEDseq**

Version: 1.4.2

Date: 2025-03-10 (this version), 2019-08-24 (original release)

Licence: GPL (>= 3)

Usage

Fits _MEDseq_ models introduced by Murphy et al. (2021) <doi:10.1111/rssa.12712>, i.e. fits mixtures of exponential-distance models for clustering longitudinal life-course sequence data via the EM/CEM algorithm.

A family of parsimonious precision parameter constraints are accommodated. So too are sampling weights. Gating covariates can be supplied via formula interfaces.

The most important function in the **MEDseq** package is: `MEDseq_fit`, for fitting the models via EM/CEM. This function requires the data to be in `"stslist"` format; the function `seqdef` is conveniently reexported from the **TraMineR** package for this purpose.

`MEDseq_control` allows supplying additional arguments which govern, among other things, controls on the initialisation of the allocations for the EM/CEM algorithm and the various model selection options.

`MEDseq_compare` is provided for conducting model selection between different results from using different covariate combinations &/or initialisation strategies, etc.

`MEDseq_stderr` is provided for computing the standard errors of the coefficients for the covariates in the gating network.

A dedicated plotting function `plot.MEDseq` exists for visualising various aspects of the results, using new methods as well as some existing methods adapted from the **TraMineR** package.

Finally, the package also contains two data sets: `biofam` and `mvad`.

Author(s)

Keefe Murphy [aut, cre], Thomas Brendan Murphy [ctb], Raffaella Piccarreta [ctb], Isobel Claire Gormley [ctb]

Maintainer: Keefe Murphy - <<keefe.murphy@mu.ie>>

References

Murphy, K., Murphy, T. B., Piccarreta, R., and Gormley, I. C. (2021). Clustering longitudinal life-course sequences using mixtures of exponential-distance models. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 184(4): 1414-1451. <doi:10.1111/rssa.12712>.

See Also

Useful links:

- https://cran.r-project.org/package=MEDseq
- Report bugs at https://github.com/Keefe-Murphy/MEDseq

Further details and examples are given in the associated vignette document:

```
vignette("MEDseq", package = "MEDseq")
```

Examples

```
# Load the MVAD data
data(mvad)
mvad$Location <- factor(apply(mvad[,5:9], 1L, function(x)
                   which(x == "yes")), labels = colnames(mvad[,5:9]))
mvad          <- list(covariates = mvad[c(3:4,10:14,87)],
                      sequences = mvad[,15:86],
                      weights = mvad[,2])
mvad.cov      <- mvad$covariates

# Create a state sequence object with the first two (summer) time points removed
states        <- c("EM", "FE", "HE", "JL", "SC", "TR")
labels        <- c("Employment", "Further Education", "Higher Education",
                   "Joblessness", "School", "Training")
mvad.seq      <- seqdef(mvad$sequences[-c(1,2)], states=states, labels=labels)

# Fit a range of unweighted models without covariates
# Only consider models with a noise component
# Supply some MEDseq_control() arguments
mod1          <- MEDseq_fit(mvad.seq, G=9:10, modtype=c("CCN", "CUN", "UCN", "UUN"),
                            algo="CEM", init.z="kmodes", criterion="icl")

# Fit a model with weights and gating covariates
# Have the probability of noise-component membership be constant
mod2          <- MEDseq_fit(mvad.seq, G=11, modtype="UUN", weights=mvad$weights,
                            gating=~ gcse5eq, covars=mvad.cov, noise.gate=FALSE)

# Examine this model and its gating network
summary(mod2, network=TRUE)
plot(mod2, "clusters")
```

`biofam` *Family life states from the Swiss Household Panel biographical survey*

Description

2000 16 year-long family life sequences built from the retrospective biographical survey carried out by the Swiss Household Panel (SHP) in 2002.

Usage

```
data(biofam)
```

Format

A data frame with 2000 rows, 16 state variables, 1 id variable and 7 covariates and 2 weights variables.

Details

The *biofam* data set was constructed by Müller et al. (2007) from the data of the retrospective biographical survey carried out by the Swiss Household Panel (SHP) in 2002.

The data set contains (in columns 10 to 25) sequences of family life states from age 15 to 30 (sequence length is 16) and a series of covariates. The sequences are a sample of 2000 sequences of those created from the SHP biographical survey. It includes only individuals who were at least 30 years old at the time of the survey. The *biofam* data set describes family life courses of 2000 individuals born between 1909 and 1972.

The states numbered from 0 to 7 are defined from the combination of five basic states, namely Living with parents (Parent), Left home (Left), Married (Marr), Having Children (Child), Divorced:

0 = "Parent"
1 = "Left"
2 = "Married"
3 = "Left+Marr"
4 = "Child"
5 = "Left+Child"
6 = "Left+Marr+Child"
7 = "Divorced"

The covariates are:

`sex`	
`birthyr`	(birth year)
`nat_1_02`	(first nationality)
`plingu02`	(language of questionnaire)
`p02r01`	(religion)
`p02r04`	(religious participation)
`cspfaj`	(father's social status)
`cspmoj`	(mother's social status)

Two additional weights variables are inserted for illustrative purpose ONLY (since `biofam` is a subsample of the original data, these weights are not adapted to the actual data):

`wp00tbgp`	(weights inflating to the Swiss population)
`wp00tbgs`	(weights respecting sample size)

Source

Swiss Household Panel https://forscenter.ch/projects/swiss-household-panel/

References

Müller, N. S., Studer, M. and Ritschard, G. (2007). Classification de parcours de vie à l'aide de l'optimal matching. In *XIVe Rencontre de la Société francophone de classification (SFC 2007)*, Paris, 5-7 septembre 2007, pp. 157-160.

Examples

```
data(biofam, package="MEDseq")

biofam         <- list(covariates = biofam[2L:9L], sequences = biofam[10L:25L] + 1L)
biofam.cov     <- biofam$covariates[,colSums(is.na(biofam$covariates)) == 0]
biofam.seq     <- seqdef(biofam$sequences,
                         states = c("P", "L", "M", "L+M", "C", "L+C", "L+M+C", "D"),
                         labels = c("Parent", "Left", "Married", "Left+Marr", "Child",
                                    "Left+Child", "Left+Marr+Child", "Divorced"))
biofam.cov$age <- 2002 - biofam.cov$birthyr
```

---

dbs *Compute the Density-based Silhouette*

Description

Computes the Density-based Silhouette for a 'soft' clustering assignment matrix.

Usage

```
dbs(z,
    ztol = 1E-100,
    weights = NULL,
    summ = c("mean", "median"),
    clusters = NULL,
    ...)
```

Arguments

z	A numeric matrix such that rows correspond to observations, columns correspond to clusters, and rows sum to 1.
ztol	A small (single, numeric, non-negative) tolerance parameter governing whether small assignment probabilities are treated instead as crisp assignments. Defaults to 1E-100.
weights	An optional numeric vector giving observation-specific weights for computing the (weighted) mean/median DBS (see summ).
summ	A single character string indicating whether the (possibly weighted) "mean" (the default) or "median" DBS should be computed.
clusters	Optional/experimental argument for giving the indicator labels of the cluster assignments. Defaults to the MAP assignment derived from z when not supplied. Note that actually supplying the MAP assignment here is slightly less efficient than the NULL default and **not** advised.
...	Catches unused arguments.

Value

A list with the following elements:

`silvals` A matrix where each row contains the cluster to which each observation belongs in the first column and the observation-specific DBS width in the second column.

`msw` Depending on the value of `summ`, either the mean or median DBS width.

`wmsw` Depending on the value of `summ`, either the weighted mean or weighted median DBS width.

Note

When calling `MEDseq_fit`, the `summ` argument can be passed via the `...` construct, in which case it governs both the `dbs` and `asw` criteria.

Author(s)

Keefe Murphy - <<keefe.murphy@mu.ie>>

References

Menardi, G. (2011). Density-based silhouette diagnostics for clustering methods. *Statistics and Computing*, 21(3): 295-308.

See Also

`MEDseq_fit`

Examples

```
# Generate a toy z matrix
z <- abs(matrix(rnorm(50), ncol=2))
z <- z/rowSums(z)

# Return the median DBS width
dbs(z, summ="median")$msw

# For real sequence data
data(mvad)

mod <- MEDseq_fit(seqdef(mvad[,17:86]), G=11, modtype="UUN", weights=mvad$weight)

dbs(mod$z, weights=mvad$weight)
```

dist_freqwH *Pairwise frequency-Weighted Hamming distance matrix for categorical data*

Description

Computes the matrix of pairwise distance using a frequency-weighted variant of the Hamming distance often used in k-modes clustering.

Usage

```
dist_freqwH(data,
            full.matrix = TRUE)
```

Arguments

data	A matrix or data frame of categorical data. Objects have to be in rows, variables in columns.
full.matrix	Logical. If TRUE (the default), the full pairwise distance matrix is returned, otherwise an object of class dist is returned, i.e. a vector containing only values from the upper triangle of the distance matrix. Objects of class dist are smaller and can be passed directly as arguments to most clustering functions.

Details

As per wKModes, the frequency weights are computed within the function and are *not* user-specified. These frequency weights are assigned on a per-feature basis and derived from the categories represented in each column of data.

Value

The whole matrix of pairwise distances if full.matrix=TRUE, otherwise the corresponding dist object.

Author(s)

Keefe Murphy - <<keefe.murphy@mu.ie>>

References

Huang, Z. (1998). Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery*, 2(3): 283-304.

See Also

wKModes, wcAggregateCases, wcSilhouetteObs

Examples

```
suppressMessages(require(WeightedCluster))
set.seed(99)
# Load the MVAD data & aggregate the state sequences
data(mvad)
agg      <- wcAggregateCases(mvad[,17:86], weights=mvad$weight)

# Create a state sequence object without the first two (summer) time points
states   <- c("EM", "FE", "HE", "JL", "SC", "TR")
labels   <- c("Employment", "Further Education", "Higher Education",
              "Joblessness", "School", "Training")
weights  <- agg$aggWeights
mvad.seq <- seqdef(mvad[agg$aggIndex, 17:86],
                   states=states, labels=labels, weights=agg$aggWeights)

# Run k-modes with weights
resW     <- wKModes(mvad.seq, 2, weights=agg$aggWeights)

# Run k-modes with additional frequency weights
resF     <- wKModes(mvad.seq, 2, weights=agg$aggWeights, freq.weighted=TRUE)

# Examine the average silhouette widths of both weighted solutions
weighted.mean(wcSilhouetteObs(seqdist(mvad.seq, method="HAM"), resW$cluster, weights), weights)
# weighted.mean(wcSilhouetteObs(seqdist(mvad.seq, method="HAM"), resF$cluster, weights), weights)
weighted.mean(wcSilhouetteObs(dist_freqwH(mvad.seq), resF$cluster, weights), weights)
```

---

get_MEDseq_results *Extract results from a MEDseq model*

Description

Utility function for extracting results of submodels from "MEDseq" objects when a range of models were run via MEDseq_fit.

Usage

```
get_MEDseq_results(x,
                   what = c("z", "MAP", "DBS", "ASW"),
                   rank = 1L,
                   criterion = c("bic", "icl", "aic", "dbs",
                                 "asw", "cv", "nec", "loglik"),
                   G = NULL,
                   modtype = NULL,
                   noise = TRUE,
                   ...)
```

Arguments

x	An object of class "MEDseq" generated by MEDseq_fit or an object of class "MEDseqCompare" generated by MEDseq_compare.
what	A character string indicating the desired results to extract.
rank	A number indicating what rank model results should be extracted from, where the rank is determined by criterion. Defaults to 1, i.e. the best model.
criterion	The criterion used to determine the ranking. Defaults to "bic".
G	Optional argument giving the number of components in the model for which results are desired. Can be supplied with or without also specifying modtype.
modtype	Optional argument giving the desired model type for which results are desired. Can be supplied with or without also specifying G.
noise	A logical indicating whether models with a noise component should be considered. Defaults to TRUE.
...	Catches unused arguments.

Details

The arguments rank and criterion are invoked when one or more of the arguments G and modtype are missing. Thus, supplying G and modtype allows rank and criterion to be bypassed entirely.

Value

The desired results extracted from the MEDseq model.

Note

Arguments to this function can be supplied to plot.MEDseq via the ... construct.

Author(s)

Keefe Murphy - <<keefe.murphy@mu.ie>>

See Also

MEDseq_fit, plot.MEDseq

Examples

```
data(biofam)
# mod <- MEDseq_fit(seqdef(biofam[10:25] + 1L), G=9:10)

# Extract the MAP clustering of the best 9-cluster model according to the asw criterion
# get_MEDseq_results(mod, what="MAP", G=9, criterion="asw")

# Extract the DBS values of the best UUN model according to the dbs criterion
# get_MEDseq_results(mod, what="DBS", modtype="UUN", criterion="dbs")

# Plot the DBS values of this same model, by passing get_MEDseq_results arguments through plot
# plot(mod, type="dbsvals", modtype="UUN", criterion="dbs")
```

MEDseq_AvePP *Average posterior probabilities of a fitted MEDseq model*

Description

Calculates the per-component average posterior probabilities of a fitted MEDseq model.

Usage

```
MEDseq_AvePP(x,
             group = TRUE)
```

Arguments

x	An object of class "MEDseq" generated by MEDseq_fit or an object of class "MEDseqCompare" generated by MEDseq_compare.
group	A logical indicating whether the average posterior probabilities should be computed *per component*. Defaults to TRUE.

Details

When group=TRUE, this function calculates AvePP, the average posterior probabilities of membership for each component for the observations assigned to that component via MAP probabilities. Otherwise, an overall measure of clustering certainty is returned.

Value

When group=TRUE, a named vector of numbers, of length equal to the number of components (G), in the range [1/G,1], such that *larger* values indicate clearer separation of the clusters. When group=FALSE, a single number in the same range is returned.

Note

This function will always return values of 1 for all components for models fitted using the "CEM" algorithm (see MEDseq_control), or models with only one component.

Author(s)

Keefe Murphy - <<keefe.murphy@mu.ie>>

References

Murphy, K., Murphy, T. B., Piccarreta, R., and Gormley, I. C. (2021). Clustering longitudinal life-course sequences using mixtures of exponential-distance models. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 184(4): 1414-1451. <doi:10.1111/rssa.12712>.

See Also

MEDseq_fit, MEDseq_control, MEDseq_entropy

Examples

```
# Load the MVAD data
data(mvad)
mvad$Location <- factor(apply(mvad[,5:9], 1L, function(x)
                 which(x == "yes")), labels = colnames(mvad[,5:9]))
mvad          <- list(covariates = mvad[c(3:4,10:14,87)],
                      sequences = mvad[,15:86],
                      weights = mvad[,2])
mvad.cov      <- mvad$covariates

# Create a state sequence object with the first two (summer) time points removed
states        <- c("EM", "FE", "HE", "JL", "SC", "TR")
labels        <- c("Employment", "Further Education", "Higher Education",
                   "Joblessness", "School", "Training")
mvad.seq      <- seqdef(mvad$sequences[-c(1,2)], states=states, labels=labels)

# Fit a model with weights and a gating covariate
# Have the probability of noise-component membership be constant
mod           <- MEDseq_fit(mvad.seq, G=11, modtype="UUN", weights=mvad$weights,
                            gating=~ gcse5eq, covars=mvad.cov, noise.gate=FALSE)

# Calculate the AvePP per component
MEDseq_AvePP(mod)

# Calculte an overall measure of clustering certainty
MEDseq_AvePP(mod, group=FALSE)
```

MEDseq_clustnames *Automatic labelling of clusters using central sequences*

Description

These functions extract names for clusters according to the SPS representation of their central sequences.

Usage

```
MEDseq_clustnames(x,
                  cluster = TRUE,
                  size = FALSE,
                  MAP = FALSE,
                  weighted = FALSE,
                  ...)

MEDseq_nameclusts(names)
```

Arguments

x	An object of class "MEDseq" generated by MEDseq_fit or an object of class "MEDseqCompare" generated by MEDseq_compare.
cluster	A logical indicating whether names should be prepended with the text "Cluster g: ", where g is the cluster number. Defaults to TRUE.
size	A logical indicating whether the (typically 'soft') size of each cluster is appended to the label of each group, expressed as a percentage of the total number of observations. Defaults to FALSE.
MAP	A logical indicating whether to use the MAP classification in the computation of the size of each cluster, or the 'soft' clustering assignment probabilities given by x$z. Defaults to FALSE, but is always TRUE for models fitted by the CEM algorithm (see MEDseq_control), and is only relevant when size=TRUE. See weighted for incorporating the sampling weights (regardless of the value of MAP). The MAP argument here plays a similar role to map.size in MEDseq_meantime.
weighted	A logical indicating whether the sampling weights (if any) are used when appending the size of each cluster to the labels. Defaults to FALSE and only relevant when size=TRUE. The MAP argument here plays a similar role to wt.size in MEDseq_meantime.
...	Catches unused arguments.
names	The output of MEDseq_clustnames to be passed to the convenience function MEDseq_nameclusts (see Details).

Details

Unlike the seqclustname function from the **WeightedCluster** package which inspired these functions, MEDseq_clustnames only returns the names themselves, not the factor variable indicating cluster membership with labels given by those names. Thus, MEDseq_nameclusts is provided as a convenience function for precisely this purpose (see Examples).

Value

For MEDseq_clustnames, a character vector containing the names for each component defined by their central sequence, and optionally the cluster name (see cluster above) and cluster size (see size above). The name for the noise component, if any, will always be simply "Noise" (or "Cluster 0: Noise").

For MEDseq_nameclusts, a factor version of x$MAP with levels given by the output of MEDseq_clustnames.

Note

The main MEDseq_clustnames function is used internally by plot.MEDseq, MEDseq_meantime, MEDseq_stderr, and also other print and summary methods, where its invocation can typically controlled via a SPS logical argument. However, the optional arguments cluster, size, MAP, and weighted can only be passed through plot.MEDseq; elsewhere cluster=TRUE, size=FALSE, MAP=FALSE, and weighted=FALSE are always assumed. When invoked within plot.MEDseq, the MAP argument is renamed to soft, where MAP=!soft such that soft=TRUE by default.

Author(s)

Keefe Murphy - <<keefe.murphy@mu.ie>>

References

Murphy, K., Murphy, T. B., Piccarreta, R., and Gormley, I. C. (2021). Clustering longitudinal life-course sequences using mixtures of exponential-distance models. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 184(4): 1414-1451. <doi:10.1111/rssa.12712>.

See Also

seqformat, seqclustname, plot.MEDseq, MEDseq_meantime, MEDseq_stderr

Examples

```
# Load the MVAD data
data(mvad)
mvad$Location <- factor(apply(mvad[,5:9], 1L, function(x)
                 which(x == "yes")), labels = colnames(mvad[,5:9]))
mvad          <- list(covariates = mvad[c(3:4,10:14,87)],
                      sequences = mvad[,15:86],
                      weights = mvad[,2])
mvad.cov      <- mvad$covariates

# Create a state sequence object with the first two (summer) time points removed
states        <- c("EM", "FE", "HE", "JL", "SC", "TR")
labels        <- c("Employment", "Further Education", "Higher Education",
                   "Joblessness", "School", "Training")
mvad.seq      <- seqdef(mvad$sequences[-c(1,2)], states=states, labels=labels)

# Fit a model with weights and a gating covariate
# Have the probability of noise-component membership depend on the covariate
mod    <- MEDseq_fit(mvad.seq, G=5, modtype="UUN", weights=mvad$weights,
                     gating=~ gcse5eq, covars=mvad.cov, noise.gate=TRUE)

# Extract the names
names  <- MEDseq_clustnames(mod, cluster=FALSE, size=TRUE)

# Get the renamed MAP cluster membership indicator vector
group  <- MEDseq_nameclusts(names)

# Use the output in plots
plot(mod, type="d", soft=FALSE, weighted=FALSE, cluster=FALSE, size=TRUE, border=TRUE)
# same as:
# seqplot(mvad.seq, type="d", group=group)

# Indeed, this function is invoked by default for certain plot types
plot(mod, type="d", soft=TRUE, weighted=TRUE)
plot(mod, type="d", soft=TRUE, weighted=TRUE, SPS=FALSE)

# Invoke this function when printing the gating network coefficients
print(mod$gating, SPS=FALSE)
```

```
print(mod$gating, SPS=TRUE)

# Invoke this function in a call to MEDseq_meantime
MEDseq_meantime(mod, SPS=TRUE)

# Invoke this function in other plots
plot(mod, type="clusters", SPS=TRUE, size=TRUE)
plot(mod, type="precision", SPS=TRUE, size=TRUE, weighted=FALSE)
```

`MEDseq_compare` *Choose the best MEDseq model*

Description

Takes one or more sets of `"MEDseq"` models fitted by `MEDseq_fit` and ranks them according to a specified model selection criterion. It's possible to respect the internal ranking within each set of models, or to discard models within each set which were already deemed sub-optimal. This function can help with model selection via exhaustive or stepwise searches.

Usage

```
MEDseq_compare(...,
               criterion = c("bic", "icl", "aic",
                             "dbs", "asw", "cv", "nec"),
               pick = 10L,
               optimal.only = FALSE)

## S3 method for class 'MEDseqCompare'
print(x,
      index = seq_len(x$pick),
      rerank = FALSE,
      digits = 3L,
      maxi = length(index),
      ...)
```

Arguments

`...`	One or more objects of class `"MEDseq"` outputted by `MEDseq_fit`. All models must have been fit to the same data set. A single *named* list of such objects can also be supplied. Additionally, objects of class `"MEDseqCompare"` outputted by this very function can also be supplied here. This argument is only relevant for the `MEDseq_compare` function and will be ignored for the associated `print` function.
`criterion`	The criterion used to determine the ranking. Defaults to `"bic"`.

`pick` The (integer) number of models to be ranked and compared. Defaults to `10L`. Will be constrained by the number of models within the `"MEDseq"` objects supplied via `...` if `optimal.only` is `FALSE`, otherwise constrained simply by the number of `"MEDseq"` objects supplied. Setting `pick=Inf` is a valid way to select all models.

`optimal.only` Logical indicating whether to only rank models already deemed optimal within each `"MEDeq"` object (`TRUE`), or to allow models which were deemed suboptimal enter the final ranking (`FALSE`, the default). See `Details`.

`x`, `index`, `rerank`, `digits`, `maxi`
Arguments required for the associated `print` function:

`x` An object of class `"MEDseqCompare"` resulting from a call to `MEDseq_compare`.

`index` A logical or numeric vector giving the indices of the rows of the table of ranked models to print. This defaults to the full set of ranked models. It can be useful when the table of ranked models is large to examine a subset via this `index` argument, for display purposes. See `rerank`.

`rerank` A logical indicating whether the ranks should be recomputed when subsetting using `index`. Defaults to `FALSE`.

`digits` The number of decimal places to round model selection criteria to (defaults to `3`).

`maxi` A number specifying the maximum number of rows/models to print. Defaults to `length(index)`.

Details

The purpose of this function is to conduct model selection on `"MEDseq"` objects, fit to the same data set, with different combinations of gating network covariates or different initialisation settings.

Model selection will have already been performed in terms of choosing the optimal number of components and MEDseq model type within each supplied set of results, but `MEDseq_compare` will respect the internal ranking of models when producing the final ranking if `optimal.only` is `FALSE`: otherwise only those models already deemed optimal within each `"MEDseq"` object will be ranked.

As such if two sets of results are supplied when `optimal.only` is `FALSE`, the 1st, 2nd, and 3rd best models could all belong to the first set of results, meaning a model deemed suboptimal according to one set of covariates could be superior to one deemed optimal under another set of covariates.

Value

A list of class `"MEDseqCompare"`, for which a dedicated print function exists, containing the following elements (each of length `pick`, and ranked according to `criterion`, where appropriate):

`data` The name of the data set to which the models were fitted.

`optimal` The single optimal model (an object of class `"MEDseq"`) among those supplied, according to the chosen `criterion`.

`pick` The final number of ranked models. May be different (i.e. less than) the supplied `pick` value.

`MEDNames` The names of the supplied `"MEDseq"` objects.

modelNames	The MEDseq model names (denoting the constraints or lack thereof on the precision parameters).
G	The optimal numbers of components.
df	The numbers of estimated parameters.
iters	The numbers of EM/CEM iterations.
bic	BIC values, ranked according to criterion (not necessarily "bic").
icl	ICL values, ranked according to criterion (not necessarily "icl").
aic	AIC values, ranked according to criterion (not necessarily "aic").
dbs	(Weighted) mean/median DBS values, ranked according to criterion (not necessarily "dbs").
asw	(Weighted) mean/median ASW values, ranked according to criterion (not necessarily "asw").
cv	Cross-validated log-likelihood values, ranked according to criterion (not necessarily "cv").
nec	NEC values, ranked according to criterion (not necessarily "nec").
loglik	Maximal log-likelihood values.
gating	The gating formulas.
algo	The algorithm used for fitting the model - either "EM", "CEM", "cemEM".
equalPro	Logical indicating whether mixing proportions were constrained to be equal across components.
opti	The method used for estimating the central sequence(s).
weights	Logical indicating whether the given model was fitted with sampling weights.
noise	Logical indicating the presence/absence of a noise component. Only displayed if at least one of the compared models has a noise component.
noise.gate	Logical indicating whether gating covariates were allowed to influence the noise component's mixing proportion. Only printed for models with a noise component, when at least one of the compared models has gating covariates.
equalNoise	Logical indicating whether the mixing proportion of the noise component for equalPro models is also equal (TRUE) or estimated (FALSE).

Note

The criterion argument here need not comply with the criterion used for model selection within each "MEDseq" object, but be aware that a mismatch in terms of criterion *may* require the optimal model to be re-fit in order to be extracted, thereby slowing down MEDseq_compare.

If random starts had been used via init.z="random.hard" or init.z="soft.random", the optimal model may not necessarily correspond to the highest-ranking model in the presence of a criterion mismatch, due to the randomness of the initialisation.

A dedicated print function exists for objects of class "MEDseqCompare" and plot.MEDseq can also be called on objects of class "MEDseqCompare".

Author(s)

Keefe Murphy - <<keefe.murphy@mu.ie>>

References

Murphy, K., Murphy, T. B., Piccarreta, R., and Gormley, I. C. (2021). Clustering longitudinal life-course sequences using mixtures of exponential-distance models. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 184(4): 1414-1451. <doi:10.1111/rssa.12712>.

See Also

MEDseq_fit, plot.MEDseq

Examples

```
data(biofam)
seqs <- seqdef(biofam[10:25] + 1L,
               states = c("P", "L", "M", "L+M", "C",
                          "L+C", "L+M+C", "D"))
covs <- cbind(biofam[2:3], age=2002 - biofam$birthyr)

# Fit a range of models
# m1 <- MEDseq_fit(seqs, G=9:10)
# m2 <- MEDseq_fit(seqs, G=9:10, gating=~sex,       covars=covs, noise.gate=FALSE)
# m3 <- MEDseq_fit(seqs, G=9:10, gating=~age,       covars=covs, noise.gate=FALSE)
# m4 <- MEDseq_fit(seqs, G=9:10, gating=~sex + age, covars=covs, noise.gate=FALSE)

# Rank only the optimal models (according to the dbs criterion)
# Examine the best model in more detail
# (comp <- MEDseq_compare(m1, m2, m3, m4, criterion="dbs", optimal.only=TRUE))
# (best <- comp$optimal)
# (summ <- summary(best, parameters=TRUE))

# Examine all models visited, including those already deemed suboptimal
# Only print models with gating covariates & 10 components
# comp2 <- MEDseq_compare(comp, m1, m2, m3, m4, criterion="dbs", pick=Inf)
# print(comp2, index=comp2$gating != "None" & comp2$G == 10)
```

MEDseq_control *Set control values for use with MEDseq_fit*

Description

Supplies a list of arguments (with defaults) for use with MEDseq_fit.

Usage

```
MEDseq_control(algo = c("EM", "CEM", "cemEM"),
               init.z = c("kmedoids", "kmodes", "kmodes2", "hc",
                          "random.hard", "soft.random", "list"),
               z.list = NULL,
               unique = TRUE,
               criterion = c("bic", "icl", "aic", "dbs", "asw", "cv", "nec"),
```

```
tau0 = NULL,
noise.gate = TRUE,
random = TRUE,
dist.mat = NULL,
do.cv = FALSE,
do.nec = FALSE,
nfolds = 10L,
nstarts = 1L,
stopping = c("aitken", "relative"),
equalPro = FALSE,
equalNoise = FALSE,
tol = c(1E-05, 1E-08),
itmax = c(.Machine$integer.max, 1000L),
opti = c("mode", "medoid", "first", "GA"),
ordering = c("none", "decreasing", "increasing"),
MaxNWts = 1000L,
verbose = TRUE,
...)
```

Arguments

algo	Switch controlling whether models are fit using the "EM" (the default) or "CEM" algorithm. The option "cemEM" allows running the EM algorithm starting from convergence of the CEM algorithm.
init.z	The method used to initialise the cluster labels. All options respect the presence of sampling weights, if any. Defaults to "kmedoids". Other options include "kmodes", "kmodes2", Ward's hierarchical clustering ("hc", via hclust), "random.hard", or "soft.random" initialisation (see nstarts below), and a user-supplied "list" (see z.list below). Note that init.z="soft.random" is only available when algo="CEM". For weighted sequences, "kmedoids" is itself initialised using Ward's hierarchical clustering. The "kmodes" and "kmodes2" options both internally call the function wKModes, which *typically* uses random initial modes. Under "kmodes", the algorithm is instead initialised via the medoids of the clusters obtained from a call to hclust. The option "kmodes2" is slightly faster, by virtue of using the *random* initial medoids. However, final results are by default still subject to randomness under both options (unless set.seed is invoked), as ties for modes and cluster assignments are *typically* broken at random throughout the algorithm (see the random argument below, and in wKModes itself).
z.list	A user supplied list of initial cluster allocation matrices, with number of rows given by the number of observations (*including duplicates* if unique=TRUE), and numbers of columns given by the range of component numbers being considered (where columns corresponding to noise components must be included). Only relevant if init.z == "z.list". These matrices are allowed correspond to both soft or hard clusterings, and will be internally normalised so that the rows sum to 1 (or coerced always to a 'hard' matrix if algo != "EM").
unique	A logical indicating whether the model is fit only to the unique observations (defaults to TRUE). When there are covariates, this means all unique combinations

of covariate and sequence patterns, otherwise only the sequence patterns.

When `weights` *are not* supplied to `MEDseq_fit` and `unique=TRUE`, weights are given by the occurrence frequency of the corresponding sequences, and the model is then fit to the unique observations only.

When `weights` *are* supplied and `unique=TRUE`, the weights are summed for each set of duplicate observations and assigned to one retained copy of each corresponding unique sequence. Hence, observations with different weights that are otherwise duplicates are treated as duplicates and significant computational gains can be made.

In both cases, the results will be unchanged, but setting `unique` to `TRUE` can often be much faster.

`criterion` When either `G` or `modtype` is a vector, `criterion` governs how the 'best' model is determined when gathering output. Defaults to `"bic"`. Note that all criteria will be returned in any case, if possible.

`tau0` Prior mixing proportion for the noise component whenever `modtype` contains one or more of `"CCN"`, `"UCN"`, `"CUN"`, or `"UUN"`. Typically supplied as a scalar in the interval (0, 1), e.g. `0.1`, such that the same prior probability of belonging to the noise component applies to *all* observations. If `tau0` is not supplied, it will default to the inverse of the number of components (*including* the noise component). However, `tau0` can also be supplied as a vector (with length equal to the number of observations, *including duplicates* if `unique=TRUE`), which may be particularly useful when gating covariates are present and `noise.gate` is `TRUE`.

`noise.gate` A logical indicating whether gating network covariates influence the mixing proportion for the noise component, if any. Defaults to `TRUE`, but leads to greater parsimony if `FALSE`. Only relevant in the presence of a noise component (i.e. the `"CCN"`, `"UCN"`, `"CUN"`, and `"UUN"` models); only affects estimation in the presence of gating covariates.

`random` A logical governing how ties for estimated central sequence positions are handled. When `TRUE` (the default), such ties are broken at random. When `FALSE` (the implied default prior to version `1.2.0` of this package), the first candidate state is always chosen. This argument affects all `opti` options. If `verbose` is `TRUE` and there are tie-breaking operations performed, a warning message is printed once per model, regardless of the number of such operations.

Note that this argument is *also* passed to `wKModes` if `init.z` is `"kmodes"` or `"kmodes2"` and that, in certain rare cases when the `"CEM"` `algo` is invoked when `equalPro` is `TRUE` and the precision parameter(s) are somehow constrained across clusters, this argument also governs ties for cluster assignments within `MEDseq_fit` as well.

`dist.mat` An optional distance matrix to use - *for initialisation only* - when `init.z` is one of `"kmedoids"`, `"kmodes"`, or `"hc"`. Defaults to a Hamming distance matrix. This is an experimental feature and should only be tampered with by expert users.

`do.cv` A logical indicating whether cross-validated log-likelihood scores should also be computed (see `nfolds`). Defaults to `FALSE` due to significant computational burden incurred.

do.nec — A logical indicating whether the normalised entropy criterion (NEC) should also be computed (for models with more than one component). Defaults to FALSE. When TRUE, models with G=1 are fitted always.

nfolds — The number of folds to use when do.cv=TRUE. Defaults to 10.

nstarts — The number of random initialisations to use when init.z="random.hard" or init.z="soft.random". Defaults to 1. Results will be based on the random start yielding the highest estimated log-likelihood.

stopping — The criterion used to assess convergence of the EM/CEM algorithm. The default ("aitken") uses Aitken's acceleration method, otherwise the "relative" change in log-likelihood is monitored (which may be less strict).

equalPro — Logical variable indicating whether or not the mixing proportions are to be constrained to be equal in the model. Default: equalPro = FALSE. Only relevant when gating covariates are *not* supplied within MEDseq_fit, otherwise ignored. In the presence of a noise component, only the mixing proportions for the non-noise components are constrained to be equal (by default, see equalNoise), after accounting for the noise component.

equalNoise — Logical which is **only** invoked when equalPro=TRUE and gating covariates are not supplied. Under the default setting (FALSE), the mixing proportion for the noise component is estimated, and remaining mixing proportions are equal; when TRUE all components, including the noise component, have equal mixing proportions.

tol — A vector of length two giving *relative* convergence tolerances for 1) the log-likelihood of the EM/CEM algorithm, and 2) optimisation in the multinomial logistic regression in the gating network, respectively. The default is c(1e-05, 1e-08). If only one number is supplied, it is used as the tolerance in both cases.

itmax — A vector of length two giving integer limits on the number of iterations for 1) the EM/CEM algorithm, and 2) the multinomial logistic regression in the gating network, respectively. The default is c(.Machine$integer.max, 1000). This allows termination of the EM/CEM algorithm to be completely governed by tol[1]. If only one number is supplied, it is used as the iteration limit for the EM/CEM algorithm only and the other element of itmax retains its usual default.

If, for any model with gating covariates, the multinomial logistic regression in the gating network fails to converge in itmax[2] iterations at any stage of the EM/CEM algorithm, an appropriate warning will be printed, prompting the user to modify this argument.

opti — Character string indicating how central sequence parameters should be estimated. The default "mode" is exact and thus this experimental argument should only be tampered with by expert users. The option "medoid" fixes the central sequence(s) to be one of the observed sequences (like k-medoids). The other options "first" and "GA" use stochastic local search with the first-improvement and genetic algorithms, respectively, to mutate the medoid. Pre-computation of the Hamming distance matrix for the observed sequences speeds-up computation of all options other than "mode".

ordering — Experimental feature that should only be tampered with by experienced users. Allows sequences to be reordered on the basis of the column-wise entropy when opti is "first" or "GA".

`MaxNWts` The maximum allowable number of weights in the call to `multinom` for the multinomial logistic regression in the gating network. There is no intrinsic limit in the code, but increasing `MaxNWts` will probably allow fits that are very slow and time-consuming. It may be necessary to increase `MaxNWts` when categorical concomitant variables with many levels are included or the number of components is high.

`verbose` Logical indicating whether to print messages pertaining to progress to the screen during fitting. By default is `TRUE` if the session is interactive, and `FALSE` otherwise. If `FALSE`, warnings and error messages will still be printed to the screen, but everything else will be suppressed.

`...` Catches unused arguments, and also allows the optional arguments `ztol` and `summ` to be passed to `dbs` (`ztol` and `summ`) as well as the ASW computation (`summ`), and the optional `wKModes` arguments `iter.max`, `freq.weighted`, and `fast` (provided `init.z` is one of `"kmodes"` or `"kmodes2"`). In such cases, the `wKModes` argument `random` is already controlled by `random` above here.

Details

`MEDseq_control` is provided for assigning values and defaults within `MEDseq_fit`. While the `criterion` argument controls the choice of the optimal number of components and MEDseq model type (in terms of the constraints or lack thereof on the precision parameters), `MEDseq_compare` is provided for choosing between fits with different combinations of covariates or different initialisation settings.

Value

A named list in which the names are the names of the arguments and the values are the values supplied to the arguments.

Author(s)

Keefe Murphy - <<keefe.murphy@mu.ie>>

References

Murphy, K., Murphy, T. B., Piccarreta, R., and Gormley, I. C. (2021). Clustering longitudinal life-course sequences using mixtures of exponential-distance models. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 184(4): 1414-1451. <doi:10.1111/rssa.12712>.

Menardi, G. (2011). Density-based silhouette diagnostics for clustering methods. *Statistics and Computing*, 21(3): 295-308.

Hoos, H. and T. Stützle (2004). *Stochastic Local Search: Foundations and Applications*. The Morgan Kaufman Series in Artificial Intelligence. San Francisco, CA, USA: Morgan Kaufman Publishers Inc.

See Also

`MEDseq_fit`, `dbs`, `wcKMedoids`, `pam`, `wKModes`, `hclust`, `seqdist`, `multinom`, `MEDseq_compare`

Examples

```
# The CC MEDseq model is almost equivalent to k-medoids when the
# CEM algorithm is employed, mixing proportions are constrained,
# and the central sequences are restricted to the observed sequences
ctrl  <- MEDseq_control(algo="CEM", equalPro=TRUE, opti="medoid", criterion="asw")

data(mvad)
# Note that ctrl must be explicitly named 'ctrl'
mod  <- MEDseq_fit(seqdef(mvad[,17:86]), G=11, modtype="CC", weights=mvad$weight, ctrl=ctrl)

# Alternatively, specify the control arguments directly
mod   <- MEDseq_fit(seqdef(mvad[,17:86]), G=11, modtype="CC", weights=mvad$weight,
                    algo="CEM", equalPro=TRUE, opti="medoid", criterion="asw")

# Note that supplying control arguments via a mix of the ... construct and the named argument
# 'control' or supplying MEDseq_control output without naming it 'control' can throw an error
```

`MEDseq_entropy` *Entropy of a fitted MEDseq model*

Description

Calculates the normalised entropy of a fitted MEDseq model.

Usage

```
MEDseq_entropy(x,
               group = FALSE)
```

Arguments

`x`	An object of class `"MEDseq"` generated by `MEDseq_fit` or an object of class `"MEDseqCompare"` generated by `MEDseq_compare`.
`group`	A logical (defaults to `FALSE`) indicating whether component-specific average entropies should be returned instead.

Details

When `group` is `FALSE`, this function calculates the normalised entropy via

$$H = -\frac{1}{n \log(G)} \sum_{i=1}^{n} \sum_{g=1}^{G} \hat{z}_{ig} \log(\hat{z}_{ig})$$

, where n and G are the sample size and number of components, respectively, and $\hat{z}_{ig}$ is the estimated posterior probability at convergence that observation i belongs to component g.

When `group` is `TRUE`,

$$H_i = -\frac{1}{\log(G)} \sum_{g=1}^{G} \hat{z}_{ig} \log(\hat{z}_{ig})$$

is computed for each observation and averaged according to the MAP classification.

Value

When `group` is `FALSE`, a single number, given by $1 - H$, in the range [0,1], such that *larger* values indicate clearer separation of the clusters. Otherwise, a vector of length `G` containing the per-component averages of the observation-specific entropies is returned.

Note

This function will always return a normalised entropy of `1` for models fitted using the `"CEM"` algorithm (see `MEDseq_control`), or models with only one component.

Author(s)

Keefe Murphy - <<keefe.murphy@mu.ie>>

References

Murphy, K., Murphy, T. B., Piccarreta, R., and Gormley, I. C. (2021). Clustering longitudinal life-course sequences using mixtures of exponential-distance models. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 184(4): 1414-1451. <doi:10.1111/rssa.12712>.

See Also

`MEDseq_fit`, `MEDseq_control`, `MEDseq_AvePP`

Examples

```
# Load the MVAD data
data(mvad)
mvad$Location <- factor(apply(mvad[,5:9], 1L, function(x)
                 which(x == "yes")), labels = colnames(mvad[,5:9]))
mvad          <- list(covariates = mvad[c(3:4,10:14,87)],
                      sequences = mvad[,15:86],
                      weights = mvad[,2])
mvad.cov      <- mvad$covariates

# Create a state sequence object with the first two (summer) time points removed
states        <- c("EM", "FE", "HE", "JL", "SC", "TR")
labels        <- c("Employment", "Further Education", "Higher Education",
                   "Joblessness", "School", "Training")
mvad.seq      <- seqdef(mvad$sequences[-c(1,2)], states=states, labels=labels)

# Fit a model with weights and a gating covariate
# Have the probability of noise-component membership be constant
mod           <- MEDseq_fit(mvad.seq, G=11, modtype="UUN", weights=mvad$weights,
                            gating=~ gcse5eq, covars=mvad.cov, noise.gate=FALSE)

# Calculate the normalised entropy
MEDseq_entropy(mod)

# Calculate the normalised entropy per cluster
MEDseq_entropy(mod, group=TRUE)
```

MEDseq_fit *MEDseq: Mixtures of Exponential-Distance Models with Covariates*

Description

Fits MEDseq models: mixtures of Exponential-Distance models with gating covariates and sampling weights. Typically used for clustering categorical/longitudinal life-course sequences. Additional arguments are available via the function MEDseq_control.

Usage

```
MEDseq_fit(seqs,
           G = 1L:9L,
           modtype = c("CC", "UC", "CU", "UU",
                       "CCN", "UCN", "CUN", "UUN"),
           gating = NULL,
           weights = NULL,
           ctrl = MEDseq_control(...),
           covars = NULL,
           ...)

## S3 method for class 'MEDseq'
summary(object,
        classification = TRUE,
        parameters = FALSE,
        network = FALSE,
        SPS = FALSE,
        ...)

## S3 method for class 'MEDseq'
print(x,
      digits = 3L,
      ...)
```

Arguments

seqs	A state-sequence object of class "stslist" as created by the seqdef function in the **TraMineR** package (which is reexported by **MEDseq** for convenience). Note that the data set must have equal sequence lengths, the intervals are assumed to be evenly spaced, and neither missing values nor void values are allowed. These conditions can be checked using is.stslist and seqhasmiss.
G	A positive integer vector specifying the numbers of mixture components (clusters) to fit. Defaults to G=1:9.
modtype	A vector of character strings indicating the type of MEDseq models to be fitted, in terms of the constraints or lack thereof on the precision parameters. By default, all valid model types are fitted (except some only where G > 1 or G > 2, see

Note). The models are named "CC", "CU", "UC", "UU", "CCN", "CUN", "UCN", and "UUN". The first letter denotes whether the precision parameters are constrained/unconstrained across clusters. The second letter denotes whether the precision parameters are constrained/unconstrained across sequence positions (i.e. time points). The third letter denotes whether one of the components is constrained to have zero-precision/infinite variance. Such a noise component assumes sequences in that cluster follow a uniform distribution.

gating A formula for determining the model matrix for the multinomial logistic regression in the gating network when fixed covariates enter the mixing proportions. Defaults to ~1, i.e. no covariates. This will be ignored where G=1. Continuous, categorical, and/or ordinal covariates are allowed. Logical covariates will be coerced to factors. Interactions, transformations, and higher order terms are permitted: the latter **must** be specified explicitly using the AsIs operator (I). The specification of the LHS of the formula is ignored. Intercept terms are included by default.

weights Optional numeric vector containing observation-specific sampling weights, which are accounted for in the model fitting and other functions where applicable. All weights must be non-negative and non-missing. weights are always internally normalised to sum to the sample size. See the unique argument to MEDseq_control to see how incorporating weights also yields computational benefits. Note that weights must **always** be explicitly supplied here; it is not enough to use weights when constructing the state sequence object via seqdef (reexported by **MEDseq** for convenience). If you *are* using a weighted "stslist" state sequence object and do not specify weights, you will be prompted to explicitly specify weights=attr(seqs, "weights") for a weighted model or weights=NULL for an unweighted model.

ctrl A list of control parameters for the EM/CEM and other aspects of the algorithm. The defaults are set by a call to MEDseq_control.

covars An optional data frame (or a matrix with named columns) in which to look for the covariates in the gating network formula, if any. If not found in covars, any supplied gating covariates are taken from the environment from which MEDseq_fit is called. Try to ensure the names of variables in covars do not match any of those in seqs.

... Catches unused arguments (see MEDseq_control).

x, object, digits, classification, parameters, network, SPS
Arguments required for the print and summary functions: x and object are objects of class "MEDseq" resulting from a call to MEDseq_fit, while digits gives the number of decimal places to round to for printing purposes (defaults to 3). classification, parameters, and network are logicals which govern whether a table of the MAP classification of observations, the mixture component parameters, and the gating network coefficients are printed, respectively. SPS governs the printing of the relevant quantities in "summaryMEDseq" objects when any of classification, parameters, &/or network are TRUE (see MEDseq_clustnames and seqformat).

Details

The function effectively allows 8 different MEDseq precision parameter settings for models with or without gating network covariates. By constraining the mixing proportions to be equal (see `equalPro` in `MEDseq_control`) an extra special case is facilitated in the latter case.

While model selection in terms of choosing the optimal number of components and the MEDseq model type is performed within `MEDseq_fit`, using one of the `criterion` options within `MEDseq_control`, choosing between multiple fits with different combinations of covariates or different initialisation settings can be done by supplying objects of class `"MEDseq"` to `MEDseq_compare`.

Value

A list (of class `"MEDseq"`) with the following named entries (of which some may be missing, depending on the `criterion` employed), mostly corresponding to the chosen optimal model (as determined by the `criterion` within `MEDseq_control`):

`call` The matched call.

`data` The input data, `seqs`.

`modtype` A character string denoting the MEDseq model type at which the optimal `criterion` occurs.

`G` The optimal number of mixture components according to `criterion`.

`params` A list with the following named components:

- `theta` A matrix with `G` rows and T columns, where T is the number of sequence positions, giving the central sequences of each cluster. The mean of the noise component is not reported, as it does not contribute in any way to the likelihood. A dedicated `print` function is provided.
- `lambda` A matrix of precision parameters. Will contain `1` row if the 1st letter of `modtype` is '`C`' and `G` columns otherwise. Will contain `1` column if the 2nd letter of `modtype` is '`C`' and T columns otherwise, where T is the number of sequence positions. Precision parameter values of zero are reported for the noise component, if any. Note that values of `Inf` are also possible, corresponding to zero-variance, which is most likely under the `"UU"` or `"UUN"` models. A dedicated `print` function is provided.
- `tau` The mixing proportions: either a vector of length `G` or, if `gating` covariates were supplied, a matrix with an entry for each observation (rows) and component (columns).

`gating` An object of class `"MEDgating"` (for which dedicated `print`, `summary`, and `predict` methods exist) and either `"multinom"` or `"glm"` (only for single-component models) giving the `multinom` regression coefficients of the `gating` network. If `gating` covariates were *NOT* supplied (or the best model has just one component), this corresponds to a RHS of `~1`, otherwise the supplied `gating` formula. As such, a fitted `gating` network is always returned even in the absence of supplied covariates or clusters. If there is a noise component (and the option `noise.gate=TRUE` is invoked), its coefficients are those for the *last* component. **Users are cautioned against making inferences about statistical significance from summaries of the coefficients in the gating network. Users are instead advised to use the function** `MEDseq_stderr`.

z	The final responsibility matrix whose [i,k]-th entry is the probability that observation i belongs to the k-th component. If there is a noise component, its values are found in the *last* column.
MAP	The vector of cluster labels for the chosen model corresponding to z, i.e. max.col(z). Observations belonging to the noise component, if any, will belong to component 0.
BIC	A matrix of *all* BIC values with length{G} rows and length(modtype) columns. See Note.
ICL	A matrix of *all* ICL values with length{G} rows and length(modtype) columns. See Note.
AIC	A matrix of *all* AIC values with length{G} rows and length(modtype) columns. See Note.
DBS	A matrix of *all* (weighted) mean/median DBS values with length{G} rows and length(modtype) columns. See Note and dbs.
DBSvals	A list of lists giving the observation-specific DBS values for *all* fitted models. The first level of the list corresponds to numbers of components, the second to the MEDseq model types.
dbs	The (weighted) mean/median DBS value corresponding to the optimal model. May not necessarily be the optimal DBS.
dbsvals	Observation-specific DBS values corresponding to the optimum model, which may not be optimal in terms of DBS.
ASW	A matrix of *all* (weighted) mean/median ASW values with length{G} rows and length(modtype) columns. See Note and wcSilhouetteObs.
ASWvals	A list of lists giving the observation-specific ASW values for *all* fitted models. The first level of the list corresponds to numbers of components, the second to the MEDseq model types.
asw	The (weighted) mean/median ASW value corresponding to the optimal model. May not necessarily be the optimal ASW.
aswvals	Observation-specific ASW values corresponding to the optimum model, which may not be optimal in terms of ASW.
LOGLIK	A matrix of *all* maximal log-likelihood values with length{G} rows and length(modtype) columns. See Note.
DF	A matrix giving the numbers of estimated parameters (i.e. the number of 'used' degrees of freedom) for *all* visited models, with length{G} rows and length(modtype) columns. Subtract these numbers from the sample size to get the degrees of freedom. See Note.
ITERS	A matrix giving the total number of EM/CEM iterations for *all* visited models, with length{G} rows and length(modtype) columns. See Note.
CV	A matrix of *all* cross-validated log-likelihood values with length{G} rows and length(modtype) columns, if available. See Note and the arguments do.cv and nfolds to MEDseq_control.
NEC	A matrix of *all* NEC values with length{G} rows and length(modtype) columns, if available. See Note and the argument do.nec to MEDseq_control.

bic	The BIC value corresponding to the optimal model. May not necessarily be the optimal BIC.
icl	The ICL value corresponding to the optimal model. May not necessarily be the optimal ICL.
aic	The AIC value corresponding to the optimal model. May not necessarily be the optimal AIC.
loglik	The vector of increasing log-likelihood values for every EM/CEM iteration under the optimal model. The last element of this vector is the maximum log-likelihood achieved by the parameters returned at convergence.
df	The number of estimated parameters in the optimal model (i.e. the number of 'used' degrees of freedom). Subtract this number from the sample size to get the degrees of freedom.
iters	The total number of EM/CEM iterations for the optimal model.
cv	The cross-validated log-likelihood value corresponding to the optimal model, if available. May not necessarily be the optimal one.
nec	The NEC value corresponding to the optimal model, if available. May not necessarily be the optimal NEC.
ZS	A list of lists giving the z matrices for *all* fitted models. The first level of the list corresponds to numbers of components, the second to the MEDseq model types.
uncert	The uncertainty associated with the classification.
covars	A data frame gathering the set of covariates used in the gating network, if any. Will contain zero columns in the absence of gating covariates. Supplied gating covariates will be excluded if the optimal model has only one component. May have fewer columns than covariates supplied via the covars argument also, as only the included covariates are gathered here.

Dedicated plot, print, and summary functions exist for objects of class "MEDseq".

Note

Where BIC, ICL, AIC, DBS, ASW, LOGLIK, DF, ITERS, CV, and NEC contain NA entries, this corresponds to a model which was not run; for instance a UU model is never run for single-component models as it is equivalent to CU, while a UCN model is never run for two-component models as it is equivalent to CCN. As such, one can consider the value as not really missing, but equivalent to the corresponding value. On the other hand, -Inf represents models which were terminated due to error, for which a log-likelihood could not be estimated. These objects all inherit the class "MEDCriterion" for which dedicated print, summary, and plot methods exist. For plotting, the method is a wrapper to plot.MEDseq, with its type argument specified appropriately, which is the recommended approach. Note that all but "LOGLIK" can be used as model selection criteria via MEDseq_control.

Author(s)

Keefe Murphy - <<keefe.murphy@mu.ie>>

References

Murphy, K., Murphy, T. B., Piccarreta, R., and Gormley, I. C. (2021). Clustering longitudinal life-course sequences using mixtures of exponential-distance models. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 184(4): 1414-1451. <doi:10.1111/rssa.12712>.

See Also

seqdef (reexported by **MEDseq** for convenience), MEDseq_control, MEDseq_compare, plot.MEDseq, predict.MEDgating, MEDseq_stderr, MEDseq_clustnames, dbs, wcSilhouetteObs, seqformat, is.stslist, seqhasmiss, I

Examples

```
# Load the MVAD data
data(mvad)
mvad$Location <- factor(apply(mvad[,5:9], 1L, function(x)
                   which(x == "yes")), labels = colnames(mvad[,5:9]))
mvad          <- list(covariates = mvad[c(3:4,10:14,87)],
                      sequences = mvad[,15:86],
                      weights = mvad[,2])
mvad.cov      <- mvad$covariates

# Create a state sequence object with the first two (summer) time points removed
states        <- c("EM", "FE", "HE", "JL", "SC", "TR")
labels        <- c("Employment", "Further Education", "Higher Education",
                   "Joblessness", "School", "Training")
mvad.seq      <- seqdef(mvad$sequences[-c(1,2)], states=states, labels=labels)

# Fit a range of exponential-distance models without clustering
mod0          <- MEDseq_fit(mvad.seq, G=1)


# Fit a range of unweighted mixture models without covariates
# Only consider models with a noise component
# Supply some MEDseq_control() arguments
# mod1        <- MEDseq_fit(mvad.seq, G=9:10, modtype=c("CCN", "CUN", "UCN", "UUN"),
#                           algo="CEM", init.z="kmodes", criterion="icl")

# Fit a model with weights and a gating covariate
# Have the probability of noise-component membership be constant
mod2          <- MEDseq_fit(mvad.seq, G=11, modtype="UUN", weights=mvad$weights,
                            gating=~ gcse5eq, covars=mvad.cov, noise.gate=FALSE)

# Examine this model in greater detail
summary(mod2, classification=TRUE, parameters=TRUE)
summary(mod2$gating, SPS=TRUE)
print(mod2$params$theta, SPS=TRUE)
plot(mod2, "clusters")
```

MEDseq_meantime *Compute the mean time spent in each sequence category*

Description

Computes the mean time (per cluster) spent in each sequence category (i.e. state value) for a fitted MEDseq model.

Usage

```
MEDseq_meantime(x,
                MAP = FALSE,
                weighted = TRUE,
                norm = TRUE,
                prop = FALSE,
                map.size = FALSE,
                wt.size = FALSE,
                SPS = FALSE)

## S3 method for class 'MEDseqMeanTime'
print(x,
      digits = 3L,
      ...)
```

Arguments

x	An object of class "MEDseq" generated by MEDseq_fit or an object of class "MEDseqCompare" generated by MEDseq_compare.
MAP	A logical indicating whether to use the MAP classification in the computation of the averages, or the 'soft' clustering assignment probabilities given by x$z. Defaults to FALSE, but is always TRUE for models fitted by the CEM algorithm (see MEDseq_control). See weighted for incorporating the sampling weights (regardless of the value of MAP). See map.size below.
weighted	A logical indicating whether the sampling weights (if used during model fitting) are used to compute the weighted averages. These can be used alone (when MAP is TRUE) or in conjunction with the 'soft' clustering assignment probabilities (when MAP is FALSE). Defaults to TRUE. Note that, *by default*, the first column of the output is not affected by the value of weighted (see wt.size).
norm	A logical indicating whether the mean times (outputted values after the first column) are normalised to sum to the sequence length within each cluster (defaults to TRUE). Otherwise, when FALSE, entries beyond the first column give the total (weighted) number of times a given sequence category was observed in a given cluster.
prop	A logical (defaulting to FALSE and only invoked when norm is also TRUE) which further normalises the output to give the *proportions* of time spent in each state on average instead of the absolute values.

map.size	A logical (defaulting to FALSE, unless the model was fitted by the CEM algorithm (see MEDseq_control)) which overrides MAP in the Size column (or Weighted.Size column, see wt.size) of the output, e.g. if MAP=FALSE and map.size=TRUE, the MAP classification is used to determine the cluster sizes but the soft cluster-membership probabilities are used to calculate quantities in remaining columns. Only relevant when MAP=FALSE or wt.size=TRUE.
wt.size	A logical (defaults to FALSE and only invoked when when weighted is also TRUE) which toggles whether the weights are *also* used in the computation of the cluster sizes in the first column of the output (regardless of the values of MAP or map.size).
SPS	A logical indicating whether the output should be labelled according to the state-permanence-sequence representation of the central sequences. Defaults to FALSE. See MEDseq_clustnames and seqformat.
digits	Minimum number of significant digits to be printed in values.
...	Catches unused arguments.

Details

Models with weights, covariates, &/or a noise component are also accounted for.

Value

A matrix with sequence category and cluster-specific mean times, giving clusters on the rows, corresponding cluster sizes (or weighted cluster sizes) in the first column, and sequence categories in the remaining columns.

Note

The function plot.MEDseq with the option type="mt" can be used to visualise the mean times (by cluster). However, the results displayed therein (at present) always assume norm=TRUE and prop=FALSE and assume by default that map.size=FALSE and wt.size=TRUE, while the MAP argument is renamed to soft, where MAP=!soft such that soft=TRUE by default, and weighted can also be user-specified but defaults again to TRUE.

Author(s)

Keefe Murphy - <<keefe.murphy@mu.ie>>

References

Murphy, K., Murphy, T. B., Piccarreta, R., and Gormley, I. C. (2021). Clustering longitudinal life-course sequences using mixtures of exponential-distance models. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 184(4): 1414-1451. <doi:10.1111/rssa.12712>.

See Also

MEDseq_fit, MEDseq_control, plot.MEDseq

Examples

```
data(biofam)
seqs <- seqdef(biofam[10:25] + 1L,
               states = c("P", "L", "M", "L+M", "C",
                          "L+C", "L+M+C", "D"))
mod  <- MEDseq_fit(seqs, G=10, modtype="UUN")

MEDseq_meantime(mod)
MEDseq_meantime(mod, prop=TRUE)
MEDseq_meantime(mod, map.size=TRUE)
MEDseq_meantime(mod, MAP=TRUE, norm=FALSE, SPS=TRUE)
```

MEDseq_news *Show the NEWS file*

Description

Show the NEWS file of the **MEDseq** package.

Usage

```
MEDseq_news()
```

Value

The **MEDseq** NEWS file, provided the session is interactive.

Examples

```
MEDseq_news()
```

MEDseq_stderr *MEDseq gating network standard errors*

Description

Computes standard errors of the gating network coefficients in a fitted MEDseq model using either the Weighted Likelihood Bootstrap or Jackknife methods.

Usage

```
MEDseq_stderr(mod,
              method = c("WLBS", "Jackknife"),
              N = 1000L,
              symmetric = TRUE,
              SPS = FALSE)
```

Arguments

mod	An object of class "MEDseq" generated by MEDseq_fit or an object of class "MEDseqCompare" generated by MEDseq_compare.
method	The method used to compute the standard errors (defaults to "WLBS", the Weighted Likelihood Bootstrap).
N	The (integer) number of samples to use when the "WLBS" method is employed. Defaults to 1000L. Not relevant when method="Jackknife", in which case N is always the number of observations. Must be > 1, though N being greater than or equal to the sample size is recommended under method="WLBS".
symmetric	A logical indicating whether symmetric draws from the uniform Dirichlet distribution are used for the WLBS method in the presence of existing sampling weights. Defaults to TRUE; when FALSE, the concentration parameters of the Dirichlet distribution are given by the sampling weights. Only relevant when method="WLBS" for models with existing sampling weights.
SPS	A logical indicating whether the output should be labelled according to the state-permanence-sequence representation of the central sequences. Defaults to FALSE. See MEDseq_clustnames and seqformat.

Details

A progress bar is displayed as the function iterates over the N samples. The function may take a long time to run for large N. The function terminates immediately if mod$G == 1.

Value

A list with the following two elements:

Coefficients The original matrix of estimated coefficients (coef(mod$gating)).

Std. Errors The matrix of corresponding standard error estimates.

Note

The symmetric argument is an experimental feature. More generally, caution is advised in interpreting the standard error estimates under *either* the "WLBS" **or** the "Jackknife" method when there are existing sampling weights which arise from complex/stratified sampling designs.

Author(s)

Keefe Murphy - <<keefe.murphy@mu.ie>>

References

Murphy, K., Murphy, T. B., Piccarreta, R., and Gormley, I. C. (2021). Clustering longitudinal life-course sequences using mixtures of exponential-distance models. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 184(4): 1414-1451. <doi:10.1111/rssa.12712>.

O'Hagan, A., Murphy, T. B., Scrucca, L., and Gormley, I. C. (2019). Investigation of parameter uncertainty in clustering using a Gaussian mixture model via jackknife, bootstrap and weighted likelihood bootstrap. *Computational Statistics*, 34(4): 1779-1813.

See Also

MEDseq_fit, MEDseq_clustnames, seqformat

Examples

```
# Load the MVAD data
data(mvad)
mvad$Location <- factor(apply(mvad[,5:9], 1L, function(x)
                 which(x == "yes")), labels = colnames(mvad[,5:9]))
mvad          <- list(covariates = mvad[c(3:4,10:14,87)],
                      sequences = mvad[,15:86],
                      weights = mvad[,2])
mvad.cov      <- mvad$covariates

# Create a state sequence object with the first two (summer) time points removed
states        <- c("EM", "FE", "HE", "JL", "SC", "TR")
labels        <- c("Employment", "Further Education", "Higher Education",
                   "Joblessness", "School", "Training")
mvad.seq      <- seqdef(mvad$sequences[-c(1,2)], states=states, labels=labels)

# Fit a model with weights and a gating covariate
# Have the probability of noise-component membership be constant
# mod         <- MEDseq_fit(mvad.seq, G=11, modtype="UUN", weights=mvad$weights,
#                           gating=~ gcse5eq, covars=mvad.cov, noise.gate=FALSE)

# Estimate standard errors using 100 WLBS samples
# (std        <- MEDseq_stderr(mod, N=100))
```

---

mvad *MVAD: Transition from school to work*

Description

The data comes from a study by McVicar and Anyadike-Danes on transition from school to work. The data consist of static background characteristics and a time series sequence of 72 monthly labour market activities for each of a cohort of 712 individuals in the Status Zero Survey. The individuals were followed up from July 1993 to June 1999. The monthly states are recorded in columns 15 (`Jul.93`) to 86 (`Jun.99`).

Usage

```
data(mvad)
```

Format

A data frame containing 712 rows, 72 state variables, 1 id variable and 13 covariates.

Details

States are:

employment	(EM)
FE	further education (FE)
HE	higher education (HE)
joblessness	(JL)
school	(SC)
training	(TR)

The data set contains also ids (id) and sample weights (weights) as well as the following binary covariates:

male
catholic
Belfast, N.Eastern, Southern, S.Eastern, Western (location of school, one of five Education and Library Board areas in Northern Ireland)
Grammar (type of secondary education, 1=grammar school)
funemp (father's employment status at time of survey, 1=father unemployed)
gcse5eq (qualifications gained by the end of compulsory education, 1=5+ GCSEs at grades A-C, or equivalent)
fmpr (SOC code of father's current or most recent job at time of survey, 1=SOC1 (professional, managerial or related))
livboth (living arrangements at time of first sweep of survey (June 1995), 1=living with both parents)

Note

The first two months of the observation period coincide with summer holidays from school. Hence, documented examples throughout this package extract only the states in columns 17 to 86; i.e. sequences of length 70 from Sep.93 to Jun.99.

Source

McVicar and Anyadike-Danes (2002)

References

McVicar, D. (2000). Status 0 four years on: young people and social exclusion in Northern Ireland. *Labour Market Bulletin*, 14, 114-119.

McVicar, D. and Anyadike-Danes, M. (2002). Predicting successful and unsuccessful transitions from school to work by using sequence methods. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 165(2): 317-334.

Examples

```
data(mvad, package="MEDseq")

mvad$Location <- factor(apply(mvad[,5:9], 1L, function(x)
```

```
                which(x == "yes")), labels = colnames(mvad[,5:9]))
mvad          <- list(covariates = mvad[c(3:4,10:14,87)],
                      sequences = mvad[,15:86],
                      weights = mvad[,2])
mvad.cov      <- mvad$covariates

# Create a state sequence object with the first two (summer) time points removed
states        <- c("EM", "FE", "HE", "JL", "SC", "TR")
labels        <- c("Employment", "Further Education", "Higher Education",
                   "Joblessness", "School", "Training")
mvad.seq      <- seqdef(mvad$sequences[-c(1,2)], states=states,
                        labels=labels, weights=mvad$weights)
```

`plot.MEDseq` *Plot MEDseq results*

Description

Produces a range of plots of the results of fitted `MEDseq` models.

Usage

```
## S3 method for class 'MEDseq'
plot(x,
     type = c("clusters", "central", "precision", "gating",
              "bic", "icl", "aic", "dbs", "asw", "cv",
              "nec", "LOGLIK", "dbsvals", "aswvals", "similarity",
              "uncert.bar", "uncert.profile", "loglik",
              "d", "dH", "f", "Ht", "i", "I", "ms", "mt"),
     seriated = c("observations", "both", "clusters", "none"),
     soft = NULL,
     weighted = TRUE,
     SPS = NULL,
     smeth = "TSP",
     sortv = NULL,
     subset = NULL,
     quant.scale = NULL,
     ...)
```

Arguments

`x`	An object of class `"MEDseq"` generated by `MEDseq_fit` or an object of class `"MEDseqCompare"` generated by `MEDseq_compare`.
`type`	A character string giving the type of plot requested:
	`"clusters"` Visualise the data set with sequences grouped into their respective clusters. See `seriated`. Similar to the `type="I"` plot (see below). However, `type="clusters"` always plots the hard MAP partition and is unaffected by the `soft` argument below.

"central" Visualise the central sequences (typically modal sequences, but this depends on the opti argument to MEDseq_control used during model-fitting). See seriated. The central sequence for the noise component, if any, is not shown as it doesn't contribute in any way to the likelihood. See the type="ms" option below for an alternative means of displaying the central sequences.

"precision" Visualise the precision parameters in the form of a heatmap. Values of 0 and Inf are shown in "white" and "black" respectively (see quant.scale and seriated).

"gating" Visualise the gating network, i.e. the observation index (by default) against the mixing proportions for that observation, coloured by cluster. Such plots can be produced with or without the gating network actually having had covariates included during model-fitting. See seriated, but note that this argument is only relevant for models *with* gating network covariates, provided x.axis is not supplied. The optional argument x.axis can be passed via the ... construct to change the x-axis against which mixing proportions are plotted (only advisable for models with a single gating network covariate, when x.axis is a quantity related to the gating network of the fitted model).

"bic" Plots all BIC values in a fitted MEDseq object.

"icl" Plots all ICL values in a fitted MEDseq object.

"aic" Plots all AIC values in a fitted MEDseq object.

"dbs" Plots all (weighted) mean/median DBS *criterion* values in a fitted MEDseq object.

"asw" Plots all (weighted) mean/median ASW *criterion* values in a fitted MEDseq object.

"cv" Plots all cross-validated log-likelihood values in a fitted MEDseq object.

"nec" Plots all NEC values in a fitted MEDseq object.

"LOGLIK" Plots all maximal log-likelihood values in a fitted MEDseq object. While the type options above, from "bic" though to "nec", can be used as model selection criteria in MEDseq_compare and via MEDseq_control in MEDseq_fit, "LOGLIK" cannot.

"dbsvals" Silhouette plot using observations-specific DBS values for the optimal model (coloured by cluster). See seriated.

"aswvals" Silhouette plot using observations-specific ASW values for the optimal model (coloured by cluster). See seriated.

"similarity" Produces a heatmap of the similarity matrix constructed from the x$z matrix at convergence, with observations reordered via seriated for visual clarity. The (potentially seriated) similarity matrix can also be invisibly returned.

"uncert.bar" Plot the observation-specific clustering uncertainties, if any, in the form of a bar plot. Different colours will be used to distinguish observations whose uncertainty exceeds the threshold 1/x$G.

"uncert.profile" Plot the observation-specific clustering uncertainties, if any, in the form of a profile plot.

"loglik" Plot the log-likelihood at every iteration of the EM/CEM algorithm used to fit the model.

Also available are the following options which act as wrappers to types of plots produced by the seqplot function in the **TraMineR** package. All are affected by the value of seriated and all account for the sampling weights (if any) by default (see the weighted argument and the related Note below).

Note also that all of the plot types below can be made to either work with the hard MAP partition (as per seqplot), or to use the soft cluster membership probabilities, via the soft argument below. The soft information is used by default for all but the "i" and "I" plot types, which (by default) discard this information to instead use the MAP partition: see the soft argument below for modifying this default behaviour for all of the following plot types.

"d" State distribution plots (chronograms, by cluster).

"dH" State distribution plots (chronograms, by cluster) with overlaid entropy line as per type="Ht".

"f" Sequence frequency plots (by cluster).

"Ht" Transversal entropy plots (by cluster).

"i" Selected sequence index plots (by cluster). By default, bar widths for each observation will be proportional to their weight (if any). However, this can be overruled by specifying weighted=FALSE.

"I" Whole set index plots (by cluster). This plot effectively contains almost exactly the same information as type="clusters" plots, and is similarly affected by the seriated argument, albeit shown on a by-cluster basis rather than stacked in one plot. However, bar widths for each observation will (by default) be proportional to their weight (if any), which is not the case for type="clusters" plots. However, this can be overruled by specifying weighted=FALSE.

"ms" Modal state sequence plots (by cluster). This is an alternative way of displaying the central sequences beyond the type="central" option above. Notably, this option respects arguments passed to get_MEDseq_results via the ... construct (see below), while type="central" does not. **Note**: unlike type="central", this option always plots *modal* sequences, even if another opti setting was invoked during model-fitting via MEDseq_control, in which case there may be a mismatch between the visualisation and x$params$theta. Also, like type="central", nothing is shown for the noise component by default. However, it can be shown here by specifying subset appropriately (see below), despite modal sequence for noise components not actually being estimated by the model.

"mt" Mean times plots (by cluster). This is equivalent to plotting the results of MEDseq_meantime(x, MAP=!soft, weighted=weighted, norm=TRUE, prop=FALSE, map.size=!soft, wt.size=weighted). Other options, particularly for norm=FALSE and prop=TRUE, may be added in future versions of this package. By default, bar labels taken from a suitable call to MEDseq_meantime are included, but these can be suppressed by specifying bar.labels=FALSE (see below).

seriated Switch indicating whether seriation should be used to improve the visualisation by re-ordering the "observations" within clusters (the default), the "clusters", "both", or "none". See seriate and the smeth and sortv arguments below.

The "clusters" option (and the cluster-related part of "both") is only invoked when type is one of "clusters", "central", "precision", "gating", "dbsvals", "aswvals", "similarity", "d", "dH", "f", "Ht", "i", "I", "ms", or "mt" and the model has more than one component.

Additionally, the "observations" option (and the observation-related part of "both") is only invoked when type is one of "clusters", "gating", "similarity", "i" or "I", which are also the only options for which "both" is relevant.

Though all seriated options can be specified when type is "gating", they are only invoked and relevant when the model actually contains gating network covariates and x.axis is not supplied via the ... construct.

soft

This argument is a single logical indicator which is only relevant for the "clusters", "central", and "precision" plot types, as well as the "d", "dH", "f", "Ht", "i", "I", "ms", and "mt" plot types borrowed from **TraMineR**. For plots borrowed from **TraMineR**, when soft=TRUE (the default for all but the "i" and "I" type plots) the soft cluster membership probabilities are used in a manner akin to fuzzyseqplot. Otherwise, when FALSE (the default for "i" and "I" type plots), the soft information is discarded and the hard MAP partition is used instead.

Note that soft cluster membership probabilities will not be available if x$G=1 or the model was fitted using the algo="CEM" option to MEDseq_control. Plots may still be weighted when soft is FALSE, according to the observation-specific sampling weights, when weighted=TRUE, and both arguments can also be simultaneously TRUE. Note also that type="Ht" can be used in conjunction with soft=TRUE, unlike fuzzyseqplot for which type="Ht" is not permissible. Finally, be advised that plotting may be time-consuming when soft=TRUE for "i" and "I" type plots.

Additionally, for these plots and the "clusters", "central", and "precision" types, soft is passed through to MEDseq_clustnames in the rare case where SPS=TRUE (see below) and the optional MEDseq_clustnames argument size=TRUE is invoked (again, see below). Note that soft=TRUE here corresponds to MAP=FALSE in MEDseq_clustnames.

weighted

This argument is a single logical indicator which is only relevant for the "clusters", "central", and "precision" plot types, as well as the "d", "dH", "f", "Ht", "i", "I", "ms", and "mt" plot types borrowed from **TraMineR**. For plots borrowed from **TraMineR**, when TRUE (the default), the weights (if any) are accounted for in such plots. Note that when soft is TRUE, plots will still be weighted according to the soft cluster membership probabilities; thus weighted=TRUE and soft=TRUE allows both these and the observation-specific weights to be used simultaneously (the default behaviour for both arguments).

Additionally, for these plots and the "clusters", "central", and "precision" types, weighted is passed through to MEDseq_clustnames in the rare case where SPS=TRUE (see below) and the optional MEDseq_clustnames argument size=TRUE is invoked (again, see below).

SPS

A logical indicating whether clusters should be labelled according to the state-permanence-sequence representation of their central sequence. See MEDseq_clustnames and seqformat. Defaults to TRUE for the plot types adapted from **TraMineR**, i.e. the "d", "dH", "f", "Ht", "i", "I", "ms", and "mt" type plots. The SPS ar-

gument is also relevant for the following type plots: "clusters", "central", and "precision", though SPS defaults to FALSE in those instances. Note that if SPS=TRUE for any relevant plot type, the soft and weighted arguments above are relevant if the optional MEDseq_clustnames argument size=TRUE is invoked (see below).

smeth — A character string with the name of the seriation method to be used. Defaults to "TSP". See seriate and seriation::list_seriation_methods("dist") for further details and the available methods. Only relevant when seriated != "none". When seriated == "obs" or seriated == "both", the ordering of observations can be governed by smeth or *instead* governed by the sortv argument below, but the ordering of clusters (when seriated="clusters" or seriated="both") is always governed by smeth.

sortv — A sorting method governing the ordering of observations for "clusters", "gating", "similarity", "i", or "I" type plots. Potential options include "dbs" and "asw", for sorting observations by their DBS or ASW values (if available), as well as "from.start" and "from.end" (only when type is "clusters", "i", or "I"), under which sequences are sorted by the elements of the alphabet at the successive positions starting from the start/end of the sequences (as per **TraMineR**). Only relevant if seriated is one of "observations" or "both". Note that the sortv argument overrides the setting in smeth as it pertains to the ordering of observations if sortv is supplied; otherwise sortv is NULL and smeth is invoked. Note that smeth always dictates the ordering of clusters (i.e. when seriated="clusters" or seriated="both").

Additionally, when (and only when) soft=TRUE and type="I", the additional option sortv="membership" is provided in accordance with fuzzyseqplot, on which such plots are based.

subset — An optional numeric vector giving the indices of the clusters to be plotted. For models with a noise component, values in 0:x$G are admissible, where 0 and x$G can both denote the noise component, otherwise only values in 1:x$G. Only relevant for the **TraMineR**-type plots, i.e. "d", "dH", "f", "Ht", "i", "I", "ms", and "mt" type plots. Note however, that noise components are not plotted by default for type="ms" plots as the noise component's modal sequence is not estimated by the model, so subset must be explicitly specified appropriately should you wish to see it.

quant.scale — Logical indicating whether precision parameter heatmaps should use quantiles to determine non-linear colour break-points when type="precision". This ensures each colour represents an equal proportion of the data. The behaviour of 0 or Inf values remains unchanged; only strictly-positive finite entries are affected. Heavily imbalanced values are more likely for the "UU" and "UUN" model types, thus quant.scale defaults to TRUE in those instances and FALSE otherwise. Note that quant.scale is *always* FALSE for the "CC" and "CCN" model types.

... — Catches unused arguments, and allows arguments to get_MEDseq_results to be passed when type is one of "clusters", "dbsvals", "aswvals", "similarity", "uncert.bar", "uncert.profile", "d", "dH", "f", "Ht", "i", "I", "ms", or "mt", as well as the x.axis argument when type="gating".

Also allows select additional arguments to the **TraMineR** function `seqplot` (and the functions it calls in turn) to be used for the relevant plot types (e.g. `border`, `col`, `yaxis`/`xaxis` and/or `ylab`/`xlab`, `pbarw` when `type="f"`, `serr` and `bar.labels` (*as a logical only*) when `type="mt"`, `col.entr` when `type="dH"` or `type="Ht"` (with respective implied defaults of `"black"` and `"blue"`), and `info` when `type="ms"`).

For the plot types borrowed from **TraMineR**, select additional generic graphical parameters can also be supplied (see `par`, `barplot`, and `legend`). Finally, the `cluster` and `size` arguments to `MEDseq_clustnames` can be supplied where relevant, i.e. for the plot types borrowed from **TraMineR** or when `type` is one of `"clusters"`, `"central"`, or `"precision"`.

Details

The `type` options related to model selection criteria (`"bic"` through to `"nec"`) and `"LOGLIK"` plot values for *all* fitted models in the `"MEDseq"` object `x`. The remaining `type` options plot results for the optimal model, by default. However, arguments to `get_MEDseq_results` can be passed via the `...` construct to plot corresponding results for suboptimal models in `x` when `type` is one of `"clusters"`, `"d"`, `"dH"`, `"f"`, `"Ht"`, `"i"`, `"I"`, `"ms"`, or `"mt"`. See the examples below.

Value

The visualisation according to `type` of the results of a fitted `MEDseq` model.

Note

Every `type` of plot respects the sampling weights, if any. However, those related to `seqplot` plots from **TraMineR** (`"d"`, `"dH"`, `"f"`, `"Ht"`, `"i"`, `"I"`, `"ms"`, `"mt"`) do so only when `weighted=TRUE` (the default). Further customisation of such plots is possible via the `...` construct.

For these plot types borrowed from **TraMineR**, when `weighted=TRUE`, the y-axis labels (which can be suppressed using `ylab=NA`) display cluster sizes which correspond to the output of `MEDseq_meantime(x, MAP=!soft, weighted=weighted, map.size=!soft, wt.size=weighted)` when `size=TRUE` is passed via `....` The defaults of `soft=TRUE` and `weighted=TRUE` imply defaults of `MAP=FALSE`, `weighted=TRUE`, `map.size=FALSE`, and `wt.size=TRUE`, where `wt.size=TRUE` is **NOT** the default behaviour for `MEDseq_meantime`.

Warning

The plot types borrowed from **TraMineR** may be too tall/wide to display in the preview panel. This can lead to error messages about figure margins being too large and the graphics state being invalid. The same may also be true when `type` is `"dbsvals"` or `"aswvals"`.

Author(s)

Keefe Murphy - <<keefe.murphy@mu.ie>>

References

Murphy, K., Murphy, T. B., Piccarreta, R., and Gormley, I. C. (2021). Clustering longitudinal life-course sequences using mixtures of exponential-distance models. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 184(4): 1414-1451. <doi:10.1111/rssa.12712>.

Studer, M. (2018). Divisive property-based and fuzzy clustering for sequence analysis. In G. Ritschard and M. Studer (Eds.), *Sequence Analysis and Related Approaches: Innovative Methods and Applications*, Volume 10 of *Life Course Research and Social Policies*, pp. 223-239. Cham, Switzerland: Springer.

Gabadinho, A., Ritschard, G., Mueller, N. S., and Studer, M. (2011). Analyzing and visualizing state sequences in R with **TraMineR**. *Journal of Statistical Software*, 40(4): 1-37.

See Also

MEDseq_fit, seqplot, dbs, get_MEDseq_results, seriate, list_seriation_methods, fuzzyseqplot, MEDseq_meantime, MEDseq_clustnames, seqformat, MEDseq_compare, MEDseq_control

Examples

```
# Load the MVAD data
data(mvad)
mvad$Location <- factor(apply(mvad[,5:9], 1L, function(x)
                 which(x == "yes")), labels = colnames(mvad[,5:9]))
mvad          <- list(covariates = mvad[c(3:4,10:14,87)],
                      sequences = mvad[,15:86],
                      weights = mvad[,2])
mvad.cov      <- mvad$covariates

# Create a state sequence object with the first two (summer) time points removed
states        <- c("EM", "FE", "HE", "JL", "SC", "TR")
labels        <- c("Employment", "Further Education", "Higher Education",
                   "Joblessness", "School", "Training")
mvad.seq      <- seqdef(mvad$sequences[-c(1,2)], states=states, labels=labels)

# Fit a range of exponential-distance models without clustering
mod0          <- MEDseq_fit(mvad.seq, G=1)

# Show the central sequence and precision parameters of the optimal model
plot(mod0, type="central")
plot(mod0, type="ms")
plot(mod0, type="precision")

# Fit a range of unweighted mixture models without covariates
# Only consider models with a noise component
# mod1        <- MEDseq_fit(mvad.seq, G=9:11, modtype=c("CCN", "CUN", "UCN", "UUN"))

# Plot the DBS values for all fitted models
# plot(mod1, "dbs")                 #equivalent to plot(mod1$DBS)

# Plot the clusters of the optimal model (according to the dbs criterion)
# plot(mod1, "clusters", criterion="dbs")
```

```
# Use seriation to order the observations and the clusters
# plot(mod1, "clusters", criterion="dbs", seriated="both")

# Use a different seriation method
# seriation::list_seriation_methods("dist")
# plot(mod1, "clusters", criterion="dbs", seriated="both", smeth="Spectral")

# Use the DBS values instead to sort the observations, and label the clusters
# plot(mod1, "clusters", criterion="dbs", seriated="both", sortv="dbs", SPS=TRUE, size=TRUE)

# Plot the observation-specific ASW values of the best CCN model (according to the asw criterion)
# plot(mod1, "aswvals", modtype="CCN", criterion="asw")

# Plot the similarity matrix (as a heatmap) of the best G=9 model (according to the icl criterion)
# plot(mod1, "similarity", G=9, criterion="icl")

# Fit a model with weights and gating covariates
# mod2        <- MEDseq_fit(mvad.seq, G=10, modtype="UCN", weights=mvad$weights,
#                           gating=~ fmpr + gcse5eq + livboth, covars=mvad.cov)

# Plot the central sequences & precision parameters of this model
# plot(mod2, "central")
# plot(mod2, "precision")

# Plot the clustering uncertainties in the form of a barplot
# plot(mod2, "uncert.bar")

# Plot the observation-specific DBS values
# plot(mod2, "dbsvals")

# Plot the  transversal entropies by cluster & then the state-distributions by cluster
# Note that these plots may not display properly in the preview panel
# plot(mod2, "Ht", ylab=NA)          # suppress the y-axis labels
# plot(mod2, "d", border=TRUE)       # add borders
# plot(mod2, "dH", col.entr="brown", # add colour
#      ylab=NA, border=TRUE)         # both simultaneously

# The plots above use the soft cluster membership probabilities
# Discard this information and reproduce the per-cluster state-distributions plot
# plot(mod2, "d", soft=FALSE)

# The plots above use the observation-specific sampling weights
# Discard this information and plot the mean times per state per cluster
# plot(mod2, "mt", weighted=FALSE)

# Use subset to show the "fake" modal sequence for the noise component
# plot(mod2, "ms")
# plot(mod2, "ms", subset=0:mod2$G)

# Use type="I" and subset=0 to examine the noise component, with additional legend arguments
# plot(mod2, "I", subset=0, border=TRUE, weighted=FALSE,
#      seriated="none", bty="n", cex.legend=0.75)
```

predict.MEDgating *Predictions from MEDseq gating networks*

Description

Predicts mixing proportions from MEDseq gating networks. Effectively akin to predicting from a multinomial logistic regression via multinom, although here the noise component (if any) is properly accounted for. So too are models with no gating covariates at all, or models with the equal mixing proportion constraint. Prior probabilities are returned by default.

Usage

```
## S3 method for class 'MEDgating'
predict(object,
        newdata = NULL,
        type = c("probs", "class"),
        keep.noise = TRUE,
        droplevels = FALSE,
        ...)

## S3 method for class 'MEDgating'
fitted(object,
       ...)

## S3 method for class 'MEDgating'
residuals(object,
          ...)
```

Arguments

object	An object of class "MEDgating" (typically x$gating, where x is of class "MEDseq").
newdata	A matrix or data frame of test examples. If omitted, the fitted values are used.
type	The type of output desired. The default ("probs") returns prior probabilities, while "class" returns labels indicating the most likely group *a priori*. Note that observations classified assigned the noise component (if any) are given a label of 0.
keep.noise	A logical indicating whether the output should acknowledge the noise component (if any). Defaults to TRUE; when FALSE, this column is discarded and the matrix of probabilities is renormalised accordingly.
droplevels	A logical indicating whether unseen factor levels in categorical variables within newdata should be dropped (with NA predicted in their place). Defaults to FALSE.
...	Catches unused arguments or allows the type and keep.noise arguments to be passed through fitted and the keep.noise argument to be passed through residuals.

Details

This function (unlike the `predict` method for `multinom` on which `predict.MEDgating` is based) accounts for sampling weights and the various ways of treating gating covariates, equal mixing proportion constraints, and noise components, although its `type` argument defaults to `"probs"` rather than `"class"`.

Value

The return value depends on whether `newdata` is supplied or not and whether the model includes gating covariates to begin with. When `newdata` is not supplied, the fitted values are returned (as a matrix if the model contained gating covariates, otherwise as a vector as per `x$params$tau`). If `newdata` is supplied, the output is always a matrix with the same number of rows as the `newdata`.

Author(s)

Keefe Murphy - <<keefe.murphy@mu.ie>>

References

Murphy, K., Murphy, T. B., Piccarreta, R., and Gormley, I. C. (2021). Clustering longitudinal life-course sequences using mixtures of exponential-distance models. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 184(4): 1414-1451. <doi:10.1111/rssa.12712>.

See Also

`multinom`

Examples

```
# Load the MVAD data
data(mvad)
mvad$Location <- factor(apply(mvad[,5:9], 1L, function(x)
                 which(x == "yes")), labels = colnames(mvad[,5:9]))
mvad          <- list(covariates = mvad[c(3:4,10:14,87)],
                      sequences = mvad[,15:86],
                      weights = mvad[,2])
mvad.cov      <- mvad$covariates

# Create a state sequence object with the first two (summer) time points removed
states        <- c("EM", "FE", "HE", "JL", "SC", "TR")
labels        <- c("Employment", "Further Education", "Higher Education",
                   "Joblessness", "School", "Training")
mvad.seq      <- seqdef(mvad$sequences[-c(1,2)], states=states, labels=labels)

# Fit a model with weights and a gating covariate
# Have the probability of noise-component membership be constant
mod    <- MEDseq_fit(mvad.seq, G=11, modtype="UUN", weights=mvad$weights,
                     gating=~ gcse5eq, covars=mvad.cov, noise.gate=FALSE)
(preds <- predict(mod$gating, newdata=mvad.cov[1:5,]))

# Note that the predictions are not the same as the multinom predict method
```

```
# in this instance, owing to the invocation of noise.gate=FALSE above
mod2   <- mod
class(mod2$gating) <- c("multinom", "nnet")
predict(mod2$gating, newdata=mvad.cov[1:5,], type="probs")

# We can make this function behave in the same way by invoking keep.noise=FALSE
predict(mod$gating, keep.noise=FALSE, newdata=mvad.cov[1:5,])
```

---

wKModes *Weighted K-Modes Clustering with Tie-Breaking*

Description

Perform k-modes clustering on categorical data with observation-specific sampling weights and tie-breaking adjustments.

Usage

```
wKModes(data,
        modes,
        weights = NULL,
        iter.max = .Machine$integer.max,
        freq.weighted = FALSE,
        fast = TRUE,
        random = TRUE,
        ...)
```

Arguments

`data`	A matrix or data frame of categorical data. Objects have to be in rows, variables in columns.
`modes`	Either the number of modes or a set of initial (distinct) cluster modes (where each mode is a row and `modes` has the same number of columns as `data`). If a number, a random set of (distinct) rows in `data` is chosen as the initial modes. Note, this randomness is always present, and is not governed by `random` below.
`weights`	Optional numeric vector containing non-negative observation-specific case weights.
`iter.max`	The maximum number of iterations allowed. Defaults to `.Machine$integer.max`. The algorithm terminates when `iter.max` is reached or when the partition ceases to change between iterations.
`freq.weighted`	A logical indicating whether the usual simple-matching (Hamming) distance between objects is used, or a frequency weighted version of this distance. Defaults to `FALSE`; when `TRUE`, the frequency weights are computed within the algorithm and are *not* user-specified. Distinct from the observation-level `weights` above, the frequency weights are assigned on a per-feature basis and derived from the categories represented in each column of `data`. For convenience, the function `dist_freqwH` is provided for calculating the corresponding pairwise dissimilarity matrix for subsequent use.

`fast` A logical indicating whether a fast version of the algorithm should be applied. Defaults to `TRUE`.

`random` A logical indicating whether ties for the modal values &/or assignments are broken at random. Defaults to `TRUE`: the implied default had been `FALSE` prior to version `1.3.2` of this package, as per `klaR::kmodes` prior to version `1.7-1` (see Note). Note that when `modes` is specified as the number of modes, the algorithm is *always* randomly initialised, regardless of the specification of `random`.

Regarding the modes, ties are broken at random when `TRUE` and the first candidate state is always chosen for the mode when `FALSE`. Regarding assignments, tie-breaking is always first biased in favour of the observation's most recent cluster: regarding ties thereafter, these are broken at random when `TRUE` or the first other candidate cluster is always chosen when `FALSE`.

`...` Catches unused arguments.

Details

The k-modes algorithm (Huang, 1998) is an extension of the k-means algorithm by MacQueen (1967).

The data given by `data` is clustered by the k-modes method (Huang, 1998) which aims to partition the objects into k groups such that the distance from objects to the assigned cluster modes is minimised.

By default, the simple-matching (Hamming) distance is used to determine the dissimilarity of two objects. It is computed by counting the number of mismatches in all variables. Alternatively, this distance can be weighted by the frequencies of the categories in data, using the `freq.weighted` argument (see Huang, 1998, for details). For convenience, the function `dist_freqwH` is provided for calculating the corresponding pairwise dissimilarity matrix for subsequent use.

If an initial matrix of modes is supplied, it is possible that no object will be closest to one or more modes. In this case, fewer clusters than the number of supplied modes will be returned and a warning will be printed.

If called using `fast = TRUE`, the reassignment of the data to clusters is done for the entire data set before recomputation of the modes is done. For computational reasons, this option should be chosen for all but the most moderate of data sizes.

Value

An object of class `"wKModes"` which is a list with the following components:

`cluster` A vector of integers indicating the cluster to which each object is allocated.

`size` The number of objects in each cluster.

`modes` A matrix of cluster modes.

`withindiff` The within-cluster (weighted) simple-matching distance for each cluster.

`tot.withindiff` The total within-cluster (weighted) distance over all clusters. `tot.withindiff` can be used to guide the choice of the number of clusters, but beware of inherent randomness in the algorithm, which is liable to yield a jagged elbow plot (see examples).

`iterations` The number of iterations the algorithm reached.

`weighted` A logical indicating whether observation-level `weights` were used or not throughout the algorithm.

`freq.weighted` A logical indicating whether feature-level `freq.weights` were used or not in the computation of the distances. For convenience, the function `dist_freqwH` is provided for calculating the corresponding pairwise dissimilarity matrix for subsequent use.

`random` A logical indicating whether ties were broken at random or not throughout the algorithm.

Note

This code is adapted from the `kmodes` function in the **klaR** package. Specifically, modifications were made to allow for random tie-breaking for the modes and assignments (see `random` above) and the incorporation of observation-specific sampling `weights`, with a view to using this function as a means to initialise the allocations for MEDseq models (see the `MEDseq_control` argument `init.z` and the related options `"kmodes"` and `"kmodes2"`).

Notably, the `wKModes` function, when invoked inside `MEDseq_fit`, is used regardless of whether the weights are true sampling weights, or the weights are merely aggregation weights, or there are no weights at all. Furthermore, the `MEDseq_control` argument `random` is *also* passed to `wKModes` when it is invoked inside `MEDseq_fit`.

Update: as of version `1.7-1` of **klaR**, `klaR::kmodes` now breaks assignment ties at random only when `fast=TRUE`. It still breaks assignment ties when `fast=FALSE` and all ties for modal values in the non-random manner described above. Thus, the old behaviour of `klaR::kmodes` can be recovered by specifying `random=FALSE` here, but `random=TRUE` allows random tie-breaking for both types of ties in all situations.

Author(s)

Keefe Murphy - <<`keefe.murphy@mu.ie`>> (adapted from `klaR::kmodes`)

References

Huang, Z. (1998). Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery*, 2(3): 283-304.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In L. M. L. Cam and J. Neyman (Eds.), *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, Volume 1, June 21-July 18, 1965 and December 27 1965-January 7, 1966, Statistical Laboratory of the University of California, Berkeley, CA, USA, pp. 281-297. University of California Press.

See Also

`MEDseq_control`, `MEDseq_fit`, `dist_freqwH`, `wcAggregateCases`, `seqformat`

Examples

```
suppressMessages(require(WeightedCluster))
set.seed(99)
# Load the MVAD data & aggregate the state sequences
data(mvad)
```

```
agg      <- wcAggregateCases(mvad[,17:86], weights=mvad$weight)

# Create a state sequence object without the first two (summer) time points
states   <- c("EM", "FE", "HE", "JL", "SC", "TR")
labels   <- c("Employment", "Further Education", "Higher Education",
              "Joblessness", "School", "Training")
mvad.seq <- seqdef(mvad[agg$aggIndex, 17:86],
                   states=states, labels=labels,
                   weights=agg$aggWeights)

# Run k-modes without the weights
resX     <- wKModes(mvad.seq, 2)

# Run k-modes with the weights
resW     <- wKModes(mvad.seq, 2, weights=agg$aggWeights)

# Examine the modal sequences of both solutions
seqformat(seqdef(resX$modes), from="STS", to="SPS", compress=TRUE)
seqformat(seqdef(resW$modes), from="STS", to="SPS", compress=TRUE)

# Using tot.withindiff to choose the number of clusters

TWdiffs  <- sapply(1:5, function(k) wKModes(mvad.seq, k, weights=agg$aggWeights)$tot.withindiff)
plot(TWdiffs, type="b", xlab="K")

# Use multiple random starts to account for inherent randomness
TWDiff    <- sapply(1:5, function(k) min(replicate(10,
                    wKModes(mvad.seq, k, weights=agg$aggWeights)$tot.withindiff)))
plot(TWDiff, type="b", xlab="K")
```

Index

∗ **clustering**
MEDseq_compare, 15
MEDseq_fit, 25
wKModes, 47
∗ **control**
MEDseq_control, 18
∗ **datasets**
biofam, 4
mvad, 35
∗ **main**
MEDseq_compare, 15
MEDseq_fit, 25
plot.MEDseq, 37
∗ **package**
MEDseq-package, 2
∗ **plotting**
plot.MEDseq, 37
∗ **prediction**
predict.MEDgating, 45
∗ **utility**
dbs, 6
dist_freqwH, 8
get_MEDseq_results, 9
MEDseq_AvePP, 11
MEDseq_clustnames, 12
MEDseq_entropy, 23
MEDseq_meantime, 31
MEDseq_news, 33
MEDseq_stderr, 33
predict.MEDgating, 45

barplot, *42*
biofam, *3*, 4

dbs, 6, *22*, *28*, *30*, *43*
dist, *8*
dist_freqwH, 8, *49*

fitted.MEDgating *(*predict.MEDgating*)*, 45
formula, *26*
fuzzyseqplot, *40, 41*, *43*

get_MEDseq_results, 9, *39*, *41*, *43*

hclust, *19*, *22*

I, *26*, *30*
is.stslist, *25*, *30*

legend, *42*
list_seriation_methods, *43*

MEDseq *(*MEDseq-package*)*, 2
MEDseq-package, 2
MEDseq_AvePP, 11, *24*
MEDseq_clustnames, 12, *26*, *30*, *32*, *34, 35*, *40–43*
MEDseq_compare, *3*, *10, 11*, *13*, *15*, 15, *16, 17*, *22, 23*, *27*, *30, 31*, *34*, *37, 38*, *43*
MEDseq_control, *3*, *11–13*, 18, *22*, *24–32*, *38–40*, *43*, *49*
MEDseq_entropy, *12*, 23
MEDseq_fit, *3*, *7*, *9–13*, *15*, *18*, *20–24*, 25, *26, 27*, *31, 32*, *34, 35*, *37, 38*, *43*, *49*
MEDseq_meantime, *13, 14*, 31, *39*, *42, 43*
MEDseq_nameclusts *(*MEDseq_clustnames*)*, 12
MEDseq_news, 33
MEDseq_stderr, *3*, *13, 14*, *27*, *30*, 33
multinom, *22*, *27*, *45, 46*
mvad, *3*, 35

pam, *22*
par, *42*
plot, *29*
plot.MEDseq, *3*, *10*, *13, 14*, *17, 18*, *29, 30*, *32*, 37
predict, *27*
predict.MEDgating, *30*, 45
print.MEDseq *(*MEDseq_fit*)*, 25
print.MEDseqCompare *(*MEDseq_compare*)*, 15

print.MEDseqMeanTime *(*MEDseq_meantime*)*,
        31

residuals.MEDgating
        *(*predict.MEDgating*)*, 45

seqclustname, *13, 14*
seqdef, *3*, *25, 26*, *30*
seqdist, *22*
seqformat, *14*, *26*, *30*, *32*, *34, 35*, *40*, *43*, *49*
seqhasmiss, *25*, *30*
seqplot, *39*, *42, 43*
seriate, *39*, *41*, *43*
set.seed, *19*
summary.MEDseq *(*MEDseq_fit*)*, 25

wcAggregateCases, *8*, *49*
wcKMedoids, *22*
wcSilhouetteObs, *8*, *28*, *30*
wKModes, *8*, *19, 20*, *22*, 47