Package ‘rEMM’

July 24, 2015

Version 1.0-11

Date 2015-07-23

Title Extensible Markov Model for Modelling Temporal Relationships Between Clusters

Description Implements TRACDS (Temporal Relationships between Clusters for Data Streams), a generalization of Extensible Markov Model (EMM). TRACDS adds a temporal or order model to data stream clustering by superimposing a dynamically adapting Markov Chain. Also provides an implementation of EMM (TRACDS on top of tNN data stream clustering). Development of this package was supported in part by NSF IIS-0948893 and R21HG005912 from the National Human Genome Research Institute.

Classification/ACM G.4, H.2.8, I.5.1

URL `http://lyle.smu.edu/IDA/TRACDS/`

Depends R (>= 2.10.0), proxy (>= 0.4-7), igraph

Imports methods, stats, cluster, clusterGeneration, MASS, utils

Suggests graph, Rgraphviz, testthat

License GPL-2

NeedsCompilation yes

Author Michael Hahsler [aut, cre, cph], Margaret H. Dunham [aut, cph]

Maintainer Michael Hahsler `<mhahsler@lyle.smu.edu>`

Repository CRAN

Date/Publication 2015-07-24 07:54:05

R topics documented:

`16S` *Count Data for 16S rRNA Sequences*

Description

This data set contains count data for 16S ribosomal RNA (rRNA) sequences for the two phylogenetic classes Alphaproteobacteria and Mollicutes. The counts for 30 sequences for each class were obtained by counting the occurrence of triplets of nucleotides in windows of length 100 without any overlap. To separate sequences a row of dummy count of `NA` is used.

Usage

```
data("16S")
```

Format

`Alphaproteobacteria16S` and `Mollicutes16S` are matrices with about 449 rows and 64 (number of possible triplets) columns.

Source

The raw sequence information was obtained from the National center for biotechnology information (NCBI) website at http://www.ncbi.nih.gov/

Examples

```
data("16S")

emm <- EMM("Kullback", threshold=0.1)
build(emm, Mollicutes16S+1)

## start state for sequences have an initial state probability >0
it <- initial_transition(emm)
it[it>0]
```

---

build *Building an EMM using New Data*

Description

Add new data to an EMM.

Usage

```
build(x, newdata, ...)
```

Arguments

x	an `EMM` object. Note that the function will change the original EMM!
newdata	a vector (one observation), or a matrix or data.frame (each row is an observation)
...	further arguments. If `newdata` is a matrix or a data.frame then `verbose = TRUE` can be used to monitor the progress of building the model.

Details

`build()` performs clustering and also updates the TRACDS temporal layer.

`NA`s are handled in the data by using only the other dimensions if the data for dissimilarity computation (see package~**proxy**).

Value

A reference to the changed EMM object with the data added. Note: EMM objects store all variable data in an environment which enables us to update partial data without copying the whole object. Assignment will not create a copy! Use the provided method `copy()`.

See Also

Class `TRACDS`, `fade` and `dist` in **proxy**.

Examples

```
## load EMMTraffic data
data("EMMTraffic")
EMMTraffic

## create EMM
emm <- EMM(measure="eJaccard", threshold=0.2)

## build model using EMMTraffic data (note that the EMM object is
## changed without assignment!)
build(emm, EMMTraffic)
## same as: emm <- build(emm, EMMTraffic)

size(emm)
plot(emm)

## emm2 <- emm does not create a copy (just a reference)
## a "deep" copy is created using copy()
emm2<- copy(emm)

## convert the emm into a graph
as.igraph(emm)
```

`cluster` *Data stream clustering with tNN*

Description

Cluster new data into an existing tNN object.

Usage

```
cluster(x, newdata, ...)
```

Arguments

`x`	a `tNN` object. Note that this function canges the original object!
`newdata`	a vector (one observation), or a matrix or data.frame (each row is an observation).
`...`	further arguments like `verbose`.

Details

`cluster()` implements tNN clustering The dissimilarity between the new observation and the centers of the clusters is calculated. The new observation is assigned to the closest cluster if the dissimilarity value is smaller than the threshold (for the state). If no such state exists, a new state is created for the observation. This simple clustering algorithm is called nearest neighbor threshold nearest neighbor (threshold NN).

NAs are handled in the data by using only the other dimensions if the data for dissimilarity computation (see package~**proxy**).

The clusters which the data points in the last `cluster()` operation where assigned to can be retrieved using the method `last_clustering()`.

Value

A reference to the changed tNN object with the data added. Note: tNN objects store all variable data in an environment which enables us to update partial data without copying the whole object. Assignment will not create a copy! Use the provided method `copy()`.

See Also

Class `tNN`, `fade` and `dist` in **proxy**.

Examples

```
## load EMMTraffic data
data(EMMTraffic)

## create empty clustering
tnn <- tNN(th=0.2, measure="eJaccard")
tnn

## cluster some data
cluster(tnn, EMMTraffic)
tnn

## what clusters were the data points assigned to?
last_clustering(tnn)

## plot the clustering as a scatterplot matrix of the cluster centers
plot(tnn)
```

`combine` *Combining EMM Objects*

Description

Combines two or more EMMs into a single object.

Usage

```
## S4 method for signature 'EMM'
c(x, ..., copy=TRUE, recursive = FALSE)
```

Arguments

`x`	first `EMM` object. Note that this object will be changed by the function if `copy=FALSE`!
`...`	further objects of the same class as `x` to be combined.
`copy`	a logical. Copy `x` first? Otherwise `x` will be changed!
`recursive`	a logical. If `recursive=TRUE`, the function recursively descends through lists combining all their elements into a vector.

Value

Returns invisibly an object of the same class as `EMM`.

See Also

`EMM-class`,

Examples

```
data("16S")

## create two EMMs for different data
emm1 <- EMM("Kullback", threshold=0.1, data=Mollicutes16S+1)
emm2 <- EMM("Kullback", threshold=0.1, data=Alphaproteobacteria16S+1)

## combine the two EMMs
emm12 <- c(emm1, emm2)
## this is the same as:
## emm12 <- copy(emm1)
## c(emm12, emm2, copy=FALSE)

## recluster states so similar states in the to EMMs will be merged
emm12r <- recluster_tNN(emm12)

op <- par(mfrow = c(1, 2), pty = "s")
plot(emm12, method="graph", main="Two EMMs")
plot(emm12r, method="graph", main="Two EMMs (reclustered)")
par(op)
```

`Derwent` *Derwent Catchment Data*

Description

Data set with flow readings (in cubic meter per second) for six catchments of in the vicinity of the Derwent river in the northern UK. The data was collected daily from November 1, 1971 – January 31, 1977. The catchments are Long Bridge, Matlock Bath, Chat Sworth, What Stand Well, Ashford (Wye) and Wind Field Park (Amber).

The owner of the data is the Ridings Area Office of the Environment Agency North-East, UK.

Usage

```
data(Derwent)
```

Format

A matrix of size 1918 days times 6 catchments.

Source

UK National River Flow Archive (NRFA), http://www.ceh.ac.uk/data/nrfa/

The owner of the data is the Ridings Area Office of the Environment Agency North-East, UK.

References

Wikipedia, River Derwent, Yorkshire, http://en.wikipedia.org/wiki/River_Derwent,_Yorkshire

Wikipedia, River Wye, Derbyshire, http://en.wikipedia.org/wiki/River_Wye,_Derbyshire

Wikipedia, River Amber, http://en.wikipedia.org/wiki/River_Amber

Examples

```
data(Derwent)

i <- 1
plot(Derwent[,i], type="l", main=colnames(Derwent[i]), ylab="Gauged Flows")
```

`EMM` *Creator for Class "EMM"*

Description

Create a new object of class `"EMM"`.

Usage

```
EMM(threshold = 0.2, measure = "euclidean", distFun = NULL,
    centroids = identical(tolower(measure), "euclidean"),
    lambda = 0, data = NULL)
```

Arguments

`threshold`	Object of class `"numeric"` with the dissimilarity threshold used by the clustering algorithm for assigning a new observation to existing clusters.
`measure`	Object of class `"character"` containing the name of the dissimilarity measure used (see `dist` in **proxy** for available measures).
`distFun`	Specify a function passed on as method to `dist` in **proxy** (see `dist` in **proxy**). The character string passed on as `measure` will be used as the measure's name.

centroids	Object of class "logical" indicating if centroids are used for clusters. If FALSE, pseudo medians (first observation of a cluster) are used to represent a cluster.
lambda	Object of class "numeric" specifying the rate for fading.
data	Initial data to build the EMM. This just calls build on the new EMM.

Value

An object of class "EMM".

See Also

EMM-class

Examples

```
## load EMMTraffic data
data(EMMTraffic)

## create empty EMM
emm <- EMM(threshold=0.2, measure="eJaccard", lambda=.1)
emm

## cluster some data
build(emm, EMMTraffic)
emm

## what clusters were the data points assigned to?
last_clustering(emm)

## plot the clustering as a graph
plot(emm)
```

EMM-class *Class "EMM"*

Description

This class represents the extensible Markov Model. It consists of a simple data stream clustering algorithm (class "tNN") and a temporal layer (class "TRACDS").

Objects from the Class

Objects can be created using the creator function EMM or by directly calling new("EMM", ...). Most slots for the extended classes can be used as parameters for EMM.

Slots

The slots are described in corresponding the extended classes (see section Extends).

Extends

Class "`tNN`", directly. Class "`TRACDS`", directly.

Methods

copy `signature(x = "EMM")`: Make a copy of the EMM object. Making explicit copies is necessary since the subclasses store information in environments which are not copied for regular assignements.

size `signature(x = "EMM")`: Returns the size of the EMM (number of clusters/states).

References

M.H. Dunham, Y. Meng, J. Huang (2004): Extensible Markov Model, In: *ICDM '04: Proceedings of the Fourth IEEE International Conference on Data Mining,* pp. 371–374.

See Also

`build`, `fade`, `merge_clusters`, `plot`, `prune`, `rare_clusters`, `rare_transitions`, `remove_clusters`, `remove_transitions`, `remove_selftransitions`, `recluster`, and `score`.

`EMMsim` *Synthetic Data to Demonstrate EMMs*

Description

A simulated data set with four clusters in R^2. Each cluster is represented by a bivariate normally distributed random variable.

μ are the coordinates of the means of the distributions and Σ contains the covariance matrices. Two data stream are created using a fixed sequence $< 1, 2, 1, 3, 4 >$ through the four clusters. For the training data, the sequence is repeated 40 times (200 data points) and for the test data five times (25 data points).

The code to generate the data is shown in the Examples section below.

Usage

```
data(EMMsim)
```

Format

`EMMsim_train` and `EMMsim_test` are matrices containing the data.

`EMMsim_sequence_train` and `EMMsim_sequence_test` contain the sequence of the data through the four clusters.

Examples

```
## the data was generated by
## Not run:
set.seed(1234)

## simulated data
mu <- cbind(
    x = c(0, 0.2,1,0.9),
    y = c(0, 0.7,1,0.2)
)

sd_rho <- cbind(
    x = c(0.2, 0.15, 0.05, 0.02),
    y = c(0.1, 0.04, 0.03, 0.05),
    rho = c(0, 0.7, 0.3, -0.4)
)

Sigma <- lapply(1:nrow(sd_rho), FUN = function(i) rbind(
        c(sd_rho[i,"x"]^2, sd_rho[i,"rho"]*sd_rho[i,"x"]*sd_rho[i,"y"]),
        c(sd_rho[i,"rho"]*sd_rho[i,"x"]*sd_rho[i,"y"], sd_rho[i,"y"]^2)))


sequence <- c(1,2,1,3,4)

EMMsim_sequence_train <- rep(sequence, 40)
EMMsim_sequence_test <- rep(sequence, 5)

library("MASS")
EMMsim_train <- t(sapply(EMMsim_sequence_train, FUN = function(i)
        mvrnorm(1, mu=mu[i,], Sigma=Sigma[[i]])))
EMMsim_test <- t(sapply(rep(EMMsim_sequence_test), FUN = function(i)
        mvrnorm(1, mu=mu[i,], Sigma=Sigma[[i]])))

## End(Not run)
```

EMMTraffic *Hypothetical Traffic Data Set for EMM*

Description

Each observation in this hypothetical data set is a vector of seven values obtained from sensors located at specific points on roads. Each sensor collects a count of the number of vehicles which have crossed this sensor in the preceding time interval.

Usage

```
data(EMMTraffic)
```

Format

A matrix with 12 observations (rows).

References

M.H. Dunham, Y. Meng, J. Huang (2004): Extensible Markov Model, In: *ICDM '04: Proceedings of the Fourth IEEE International Conference on Data Mining,* pp. 371–374.

Examples

```
data(EMMTraffic)
EMMTraffic
```

`fade` *Fading Cluster Structure and EMM Layer*

Description

Reduces the weight of old observations in the data stream. `build` has a learning rate parameter `lambda`. If this parameter is set, `build` automatically fades all counts before a new data point is added. The second mechanism is to explicitly call the function~`fade` whenever fading is needed. This has the advantage that the overhead of manipulating all counts in the EMM can be reduced and that fading can be used in a more flexible manner. For example, if the data points are arriving at an irregular rate, `fade` could be called at regular time intervals (e.g., every second).

Usage

```
fade(x, t, lambda)
```

Arguments

`x`	an object of class `"EMM"`. Note that this function will change `x`.
`t`	number of time intervals (if missing 1 is used)
`lambda`	learning rate. If `lambda` is missing, the learning rate specified for the EMM is used.

Details

Old data points are faded by using a weight. We define the weight for data that is t timesteps in the past by the following strictly decreasing function:

$$w_t = 2^{-\lambda t}$$

Since the weight is multiplicative, it can be applied iteratively by multiplying at each time step all counts by $2^{-\lambda}$. For the clustering algorithm the weight of the clusters (number of data points assigned to the cluster) is faded. For the EMM the initial count vector and all transition counts are faded.

Value

Returns a reference to the changed object `x`.

See Also

`EMM` and `build`

Examples

```
data("EMMTraffic")

## For the example we use a very high learning rate
## this calls fade after each new data point
emm_l <- EMM(measure="eJaccard", threshold=0.2, lambda = 1)
build(emm_l, EMMTraffic)

## build a regular EMM for comparison
emm <- EMM(measure="eJaccard", threshold=0.2)
build(emm, EMMTraffic)

## compare the transition matrix
transition_matrix(emm)
transition_matrix(emm_l)

## compare graphs
op <- par(mfrow = c(1, 2), pty = "m")
plot(emm, main = "regular EMM")
plot(emm_l, main = "EMM with high learning rate")
par(op)
```

`find_clusters` *Find the EMM State/Cluster for an Observation*

Description

Finds the cluster and thus the EMM states for observations.

Usage

```
## S4 method for signature 'tNN,matrix'
find_clusters(x, newdata, match_cluster=c("exact", "nn"), dist = FALSE)
```

Arguments

`x`	an EMM object.
`newdata`	a matrix/data.frame with observations.
`match_cluster`	find exact or nearest neighbor (nn) cluster/state. If a number is supplied then the threshold times this number is used for exact matching.
`dist`	also report the distance to the chosen cluster/state (as a data.frame).

Value

Returns the name of the matching clusters/states or a data.frame with columns "state" and "dist" if `dist=TRUE`.

See Also

`EMM` and `tNN`

Examples

```
data("EMMTraffic")
emm <- EMM(measure="eJaccard", threshold=0.2)
emm <- build(emm, EMMTraffic)

find_clusters(emm, EMMTraffic)
find_clusters(emm, EMMTraffic, dist=TRUE)

## add noise to the data
set.seed(1234)
newdata <- sapply(EMMTraffic, jitter, amount=15)
## default is exact match
find_clusters(emm, newdata, dist=TRUE)
## match with nearest neighbor
find_clusters(emm, newdata, match_cluster="nn", dist=TRUE)
## exact match only if within .5 times threshold
find_clusters(emm, newdata, match_cluster=.5, dist=TRUE)
## exact match only if within 2 times threshold
find_clusters(emm, newdata, match_cluster=2, dist=TRUE)
```

`merge_clusters` *Merge States of an EMM*

Description

Merge several clusters/states of an EMM into a single cluster/state.

Usage

```
## S4 method for signature 'EMM,character'
merge_clusters(x, to_merge, clustering = FALSE, new_center = NULL, copy=TRUE)
```

Arguments

`x`	an `"EMM"` object. Note that the function will change this EMM!
`to_merge`	vector of names of the states/clusters to merge. The name of the first state in `to_merge` is used as the name for the new state representing the merged states.
`clustering`	is `to_merge` a vector with cluster assignments as created by a clustering algorithm?

new_center	supply new centers for the merged clusters. New centroids are automatically computed. If (pseudo) medoids are used, new medoids should be supplied. If none is supplied, the medoid of the cluster in to_merge which has the most assigned observations is used as the new medoid (warning: this is probably not a good medoid!)
copy	logical; make a copy of x before reclustering? Otherwise the function will change x!

Value

Returns the changed EMM with the states/clusters merged invisibly. If copy=FALSE then it returns a reference to the changes object passed as x.

Examples

```
data("EMMTraffic")
emm <- EMM(measure="eJaccard", threshold=0.2)
build(emm, EMMTraffic)
states(emm)

## create a new emm with states 1-3 merged
emm_m123 <- merge_clusters(emm, c("1", "2", "3"))
states(emm_m123)
```

---

plot *Visualize EMM Objects*

Description

Visualize EMM objects.

Usage

```
## S4 method for signature 'EMM,missing'
plot(x, y, method=c("igraph", "interactive",
    "graph", "MDS", "cluster_counts", "transition_counts"), data = NULL,
    parameter=NULL, ...)
```

Arguments

x	an EMM object.
y	unused (just for compatibility with the generic for plot in **graphics**)
method	see details section.
data	Project the state centers onto these data. Points which do not belong to any cluster are shown in blue.
parameter	a list of parameters for plotting.
...	further arguments passed on to plot.default.

Details

There are several methods for plotting:

`"igraph"` produces a graph representation of the EMM using **igraph**. Additional arguments like `layout` are passed on to plot for igraph.

`"interactive"` produces an interactive graph representation of the EMM (using **igraph**). Arguments suitable for `plot` in **igraph** can be passed on as `...`.

`"graph"` produces a graph representation of the EMM using **Rgraphviz**. If **Rgraphviz** is not installed/available then the method reverts to `"igraph"`.

`"MDS"` projects the cluster centers into 2-dimensional space.

`"cluster_counts"` produces a barplot for cluster counts.

`"transition_counts"` produces a barplot for transition counts.

The following plotting parameters are currently supported (by some of the visualizations):

state_counts represent state counts by vertex size? (default: `TRUE`)

arrow_width represent transition counts/probabilities by arrow width? (default: `TRUE`)

arrows use `"counts"` or `"probabilities"` for arrow width. (default: "counts")

arrow_width_multiplier, state_size_multiplier Controls the variation of vertex sizes and edge widths (default: 1).

add_labels add labels for centers (n/a for `type="graph"`).

cluster_labels cluster labels to use instead of 1,2,....

mark_clusters Use different markers for points depending on the state they belong to (only available for MDS when `data` is specified).

draw_threshold draw a circle around state centers to indicate the area in which points are assigned to the cluster (experimental, only available for MDS when `data` is specified).

mark_states, mark_state_color a vector of state names to be marked and the color(s) used for marking (default: red).

mark_transitions, mark_transitions_color a vector of transition names in the format "3->2" to be marked and the color(s) used for marking (default: red).

For some plots (e.g., `"igraph"`) `...` is passed on to the primitive plotting function and can be used to change the plot (colors, etc.) See `? igraph.plotting`. For `"graph"` the two special parameters "nAttrs" and "eAttrs" for node and edge attributes can be used.

See Also

`EMM`

Examples

```
data("EMMTraffic")
emm <- EMM(threshold=0.2, measure="eJaccard", data=EMMTraffic)

op <- par(mfrow = c(2, 2), pty = "s")
plot(emm, main="Graph (plain)",
    parameter=list(cluster_counts=FALSE, arrow_width=FALSE))
plot(emm, main="Graph")
plot(emm, method="MDS", main="MDS projection",
    xlim=c(-0.5,0.5), ylim= c(-0.5,0.5))
plot(emm, method="MDS", data = EMMTraffic,
main = "Projection of centers on data")
par(op)

## Example to create a fixed layout for igraph
g <- as.igraph(emm)
l <- layout.star(g)
plot(emm, layout=l)
```

---

predict *Predict a Future State*

Description

Predict a state or the probability distribution over states in n time steps.

Usage

```
## S4 method for signature 'TRACDS'
predict(object, current_state = NULL, n=1,
probabilities = FALSE, randomized = FALSE, prior=FALSE)
```

Arguments

`object`	an `"EMM"`/`"TRACDS"` object.
`current_state`	use a specified current state. If `NULL`, the EMM's current state is used.
`n`	number of time steps.
`probabilities`	if `TRUE`, instead of the predicted state, the probability distribution is returned.
`randomized`	if `TRUE`, the predicted state is choosen randomly with a selection probability proportional to its transition probability
`prior`	add one to each transition count. This is equal to starting with a uniform prior for the transition count distribution, i.e. initially all transitions are equally likely. It also prevents the product of probabilities to be zero if a transition was never observed.

Details

Prediction is done using A^n where A is the transition probability matrix maintained by the EMM. Random tie-breaking is used.

Value

The name of the predicted state or a vector with the probability distribution over all states.

See Also

transition_matrix

Examples

```
data("EMMTraffic")
emm <- EMM(measure="eJaccard", threshold=0.2)
emm <- build(emm, EMMTraffic)

#plot(emm) ## plot graph

## Predict state starting an state 1 after 1, 2 and 100 time intervals
## Note, state 7 is an absorbing state.
predict(emm, n=1, current_state="1")
predict(emm, n=2, current_state="1")
predict(emm, n=100, current_state="1")

## Get probability distribution
predict(emm, n=2, current_state="1", probabilities = TRUE)
```

---

prune *Prune States and/or Transitions*

Description

Simplifies an EMM and/or the clustering by removing all clusters/states and/or transitions which have a count of equal or smaller than a given threshold.

Usage

```
## S4 method for signature 'EMM'
prune(x, count_threshold, clusters = TRUE, transitions = FALSE,
    copy = TRUE, compact = TRUE)

rare_clusters(x, count_threshold, ...)
rare_transitions(x, count_threshold, ...)
```

Arguments

x	an object of class "EMM"
count_threshold	all states/edges with a count of less or equal to the threshold are removed from the model.
clusters	logical; prune clusters?
transitions	logical; prune transitions?
copy	logical; make a copy of x before reclustering? Otherwise the function will change x!

`compact`	logical; tries make the data structure used for the temporal model more compact after pruning.
`...`	further arguments (currently not used).

Value

`prune` returns invisibly an object of class `EMM`. If `copy=FALSE` then it returns a reference to the changes object passed as `x`.

`rare_clusters` returns a vector of names of rare clusters.

`rare_transitions` returns a data.frame of rare transitions.

See Also

`remove_transitions`, `remove_clusters`, `compact`

Examples

```
data("EMMTraffic")

## For the example we use a very high learning rate
emm_l <- EMM(threshold=0.2, measure="eJaccard", lambda = 1)
build(emm_l, EMMTraffic)

## show state counts and transition counts
cluster_counts(emm_l)
transition_matrix(emm_l, type="counts")

## rare state/transitions
rare_clusters(emm_l, count_threshold=0.1)
rare_transitions(emm_l, count_threshold=0.1)

## remove all states with a threshold of 0.1
emm_lr <- prune(emm_l, count_threshold=0.1)

## compare graphs
op <- par(mfrow = c(1, 2), pty = "m")
plot(emm_l, main = "EMM with high learning rate")
plot(emm_lr, main = "Simplified EMM")
par(op)
```

`recluster` *Reclustering EMM states*

Description

Use various clustering methods to recluster states/clusters in an EMM. The centers of the clusters in the EMM object are used as data points by the reclustering algorithm. States/centers put by reclustering into the same cluster are merged to produce a new reclustered EMM.

Usage

```
## S4 method for signature 'EMM'
recluster_hclust(x, k=NULL, h=NULL, method="average",
    ...,prune=NULL, copy=TRUE)
## S4 method for signature 'EMM'
recluster_kmeans(x, k, ..., prune=NULL, copy=TRUE)
## S4 method for signature 'EMM'
recluster_pam(x, k, ..., prune=NULL, copy=TRUE)
## S4 method for signature 'EMM'
recluster_reachability(x, h, ..., prune=NULL, copy=TRUE)
## S4 method for signature 'EMM'
recluster_tNN(x, threshold=NULL, ..., prune=NULL, copy=TRUE)
## S4 method for signature 'EMM'
recluster_transitions(x, threshold=NULL, ..., prune=NULL, copy=TRUE)
```

Arguments

`x`	an `"EMM"` object.
`k`	number of clusters.
`h`	heights where the dendrogram tree should be cut.
`threshold`	threshold used on the dissimilarity to join clusters for tNN. If no threshold is specified then the threshold stored in the EMM is used.
`method`	clustering method used by `hclust`.
`...`	additional arguments passed on to the clustering algorithm.
`prune`	logical; prune states with less than `prune` counts before reclustering.
`copy`	logical; make a copy of `x` before reclustering? Otherwise the function will change `x`!

Details

For `recluster_kmeans` `k` can also be a set of initial cluster centers (see argument `centers` for `kmeans` in package **stats**).

For `recluster_hclust` `k` or `h` can also be a vector. The result is then a list with several (nested) EMMs, one for each value.

For `recluster_reachability` reclusters all clusters which are reachable from each other. A cluster j is reachable from i if j's center is closer to i's center than `h` or if j is reachable by any cluster reachable by i.

For `recluster_tNN` reclusters such that two clusters with centers less than the threshold apart will be reclustered into a single cluster. This is useful, for example, after combining two models.

For `recluster_transitions` does not recluster clusters! It find groups of clusters which are overlapping (centers are less than 2 thresholds apart) and then redistributes the transition weights such that all members of one group are connected to all the members of the other group using the same weight.

Value

An object of class "EMM" or, if copy=FALSE a refernece to the changed object passed as x.

Clustering information is available as the attribute "cluster_info". The information provided depends in the clustering algorithm (see hclust, kmeans and pam).

See Also

merge_clusters, prune, kmeans, hclust, pam

Examples

```
data(EMMsim)
emm <- EMM(threshold = .2)
build(emm, EMMsim_train)

## do reclustering on a copy of the emm and plot dendrogram
emm_hc <- recluster_hclust(emm, h = 0.6)

attr(emm_hc, "cluster_info")

## compare original and clustered EMM
op <- par(mfrow = c(2, 2), pty = "m")
plot(emm, method= "MDS", main ="original EMM", data = EMMsim_train)
plot(attr(emm_hc, "cluster_info")$dendrogram)
abline(h=0.6, col="red")
plot(emm_hc, method="MDS", main ="clustered EMM", data = EMMsim_train)
plot(emm_hc, method="MDS", main ="clustered EMM")
par(op)
```

---

remove *Remove States/Clusters or Transitions from an EMM*

Description

Remove states/clusters or transitions from an EMM.

Usage

```
remove_clusters(x, to_remove, copy = TRUE)
remove_transitions(x, from, to,copy = TRUE)
remove_selftransitions(x, copy = TRUE)
```

Arguments

x	an EMM object.
to_remove	Names of states/clusters to remove.
from, to	Names of states for removing transitions. If to is missing from has to contain a matrix with two columns (from and to state names).

copy	logical; make a copy of x before reclustering? Otherwise the function will change `x`!

Details

`remove_selftransitions` removes the transitions from each state to itself.

Value

Returns a EMM with removed states/transitions. If `copy=FALSE` a reference to the object `x` with the states/transistions removed is returned.

Examples

```
data("EMMTraffic")
emm <- EMM(measure="eJaccard", threshold=0.2)
emm <- build(emm, EMMTraffic)

## remove state 3
emm_rs3 <- remove_clusters(emm, "3")

## remove transition 5->2
emm_rt52 <- remove_transitions(emm, "5", "2")

## compare EMMs
op <- par(mfrow = c(2, 2), pty = "m")
plot(emm, method = "graph", main = "original EMM")
plot(emm_rs3, method = "graph", main = "state 3 removed")
plot(emm_rt52, method = "graph", main = "transition 5->2 removed")
par(op)
```

---

score *Score a New Sequence Given an EMM*

Description

Calculates a score of how likely it is that a new sequence was generated by the same process as the sequences used to build the EMM.

Usage

```
## S4 method for signature 'EMM,matrix'
score(x, newdata, method = c("product", "log_sum", "sum",
        "log_odds", "supported_transitions", "supported_states",
        "sum_transitions",  "log_loss", "likelihood", "log_likelihood", "AIC"),
        match_cluster = "exact", prior=TRUE, normalize=TRUE,
        initial_transition = FALSE, threshold = NA)
## S4 method for signature 'EMM,EMM'
score(x, newdata, method = c("product", "log_sum", "sum",
        "supported_transitions"), match_cluster = "exact", prior=TRUE,
        initial_transition = FALSE)
```

Arguments

`x`	an `EMM` object.
`newdata`	sequence or another `EMM` object to score.
`method`	method to calculate the score (see details)
`match_cluster`	do the new observations have to fall within the threshold of the cluster (`"exact"`) or is nearest neighbor (`"nn"`) or weighted nearest neighbor (`weighted`) used? If `match_cluster` is a number n then observations need to fall within n times the clustering threshold of the cluster.
`prior`	add one to each transition count. This is equal to start with a count of one for each transition, i.e. initially all transitions are equally likely. It prevents the product of probabilities to be zero if a transition was never observed.
`normalize`	normalize the score by the length of the sequence.
`initial_transition`	include the initial transition in the computation?
`threshold`	minimum count threshold used by supported transitions and supported states.

Details

The scores for a new sequence x of length l can be computed by the following methods. For `match_cluster="exact"` or `"nn"`:

"product" Product of transition probabilities along the path of x in the model. A single missing transition (transition probability of zero) will result in a score of 0. Use `prior` to avoid this.

$$S_{\text{product}} = \sqrt[l-1]{\prod_{i=1}^{l-1} a_{s(i),s(i+1)}}$$

where $a_{s(i),s(j)}$ is the transition probability between the state representing positions i and j in the sequence.

"sum" Sum of transition probabilities along the path of x in the model.

$$S_{\text{sum}} = \frac{1}{l-1}\sum_{i=1}^{l-1} a_{s(i),s(i+1)}$$

"log_sum" Sum of the log of the transition probabilities along the path of x in the model. The ranking of the scores is equivalent to the product of probabilities, however, the calculation is more reliable since the product of probabilities might become a very small number.

A single missing transition (transition probability of zero) will result in a score of neg. infinity. Use `prior` to avoid this.

$$S_{\text{log\_sum}} = \frac{1}{l-1}\sum_{i=1}^{l-1} \log(a_{s(i),s(i+1)})$$

"supported_transitions" Fraction of transitions in the new sequence x supported (present) in the model after assigning each data point in x to a state in the model.

$$S_{\text{supported\_transitions}} = \frac{1}{l-1} \sum_{i=1}^{l-1} \mathrm{I}(a_{s(i),s(i+1)})$$

"supported_states" Fraction of points in the new sequence x for which a state (cluster) exists in the model. `match_cluster` is always `"exact"` because for `"nn"` this measure would always give 1. Note that this measure ignores transition information.

If threshold is given, then only states with a count greater than the given threshold are counted as supported.

"sum_transitions" Sum of the counts on the edges in the model on the path of sequence x normalized by the total number of transition counts in the model.

$$S_{\text{sum\_transitions}} = \frac{1}{l-1} \sum_{i=1}^{l-1} c_{s(i),s(i+1)}$$

where $c_{s(i),s(i+1)}$ is the transition count between the state representing positions i and j in the sequence.

If threshold is given, then only transitions with a count greater than the given threshold are counted as supported.

"likelihood", "log_likelihood" The likelihood of the model given the new data is the unnormalized product score (product of transition probabilities).

"log_loss" The average log loss is defined as

$$-sum(log2(a_s(i), s(i+1)))/(l-1)$$

It represents the average compression rate of the new sequence given the model.

"AIC" Akaike Information Criterion corrected for finite sample size.

$$2k - 2log(L)2k(k-1)/(n-k-1)$$

where $n = l - 1$ and k is the model complexity measured by the number of non-zero entries in the transition matrix. We use the likelihood of the model given by the proportion of supported transitions. AIC can be used for model selection where the smallest value indicates the preferred model.

where x_i represents the i-th data point in the new sequence, $a(i, j)$ is the transition probability from state i to state j in the model, $s(i)$ is the state the i-th data point (x_i) in the new sequence is assigned to. $\mathrm{I(v)}$ is an indicator function which is 0 for $v = 0$ and 1 otherwise.

For `match_cluster="weighted"`:

"product" Weighted version of the product of probabilities. The weight is the similarity between a new data point and the state in the model it is assigned to.

$$S_{\text{weighted\_product}} = \sqrt[l-1]{\prod_{i=1}^{l-1} \mathrm{simil}(x_i, s(i)) \mathrm{simil}(x_i, s(i+1)) a_{s(i),s(i+1)}}$$

"sum" Weighted version of the sum of probabilities.

$$S_{\text{weighted\_sum}} = \frac{1}{l-1}\sum_{i=1}^{l-1} \text{simil}(x_i, s(i))\text{simil}(x_i, s(i+1))a_{s(i),s(i+1)}$$

"log_sum" Weighted version of the sum of the log of probabilities.

$$S_{\text{weighted\_log\_sum}} = \frac{1}{l-1}\sum_{i=1}^{l-1} \log(\text{simil}(x_i, s(i))\text{simil}(x_i, s(i+1))a_{s(i),s(i+1)})$$

"supported_states" Same as `"supported_states"` but instead of counting the supported states, the similarity $\text{simil}(x_i, s(i))$ is used as a weight. Threshold is not implemented.

where $\text{simil}(\cdot)$ is a modified and normalized similarity function given by $\text{simil}(x, s) = 1 - \frac{1}{1+e^{-\frac{\text{d}(x,s)/t-1.5}{.2}}}$ where d is the distance measure and t is the threshold that was used to create the model.

Value

A scalar score value.

See Also

`transition` to access transition probabilities and `find_clusters` for assigning observations to states/clusters.

Examples

```
data("EMMsim")

emm <- EMM(threshold=.2)
emm <- build(emm, EMMsim_train)

score(emm, EMMsim_test) # default is method "product"


### create shuffled data (destroy temporal relationship)
### and create noisy data
test_shuffled <- EMMsim_test[sample(1:nrow(EMMsim_test)),]
test_noise <- jitter(EMMsim_test, amount=.3)

### helper for plotting
mybars <- function(...) {
  oldpar <- par(mar=c(5,10,4,2))
  ss <- rbind(...)
  barplot(ss[,ncol(ss):1], xlim=c(-1,4), beside=TRUE,
          horiz=TRUE, las=2,
          legend = rownames(ss))
  par(oldpar)
}
```

```
### compare various scores
methods <- c("product",
             "sum",
             "log_sum",
             "supported_states",
             "supported_transitions",
             "sum_transitions",
             "log_loss",
             "likelihood")

### default is exact matching
clean <- sapply(methods, FUN=function(m) score(emm, EMMsim_test, method=m))
shuffled <- sapply(methods, FUN=function(m) score(emm, test_shuffled, method=m))
noise <- sapply(methods, FUN=function(m) score(emm, test_noise, method=m))
mybars(shuffled, noise, clean)

### weighted matching is better for noisy data
clean <- sapply(methods, FUN=function(m) score(emm, EMMsim_test, method=m,
                                               match="weighted"))
shuffled <- sapply(methods, FUN=function(m) score(emm, test_shuffled, method=m,
                                                  match="weighted"))
noise <- sapply(methods, FUN=function(m) score(emm, test_noise, method=m,
                                               match="weighted"))
mybars(shuffled, noise, clean)
```

smooth_transitions *Smooths transition counts between neighboring states/clusters*

Description

Each state/cluster gets the average count if all the outgoing transitions of its neighbors (i.e., clusters which are within range x its threshold).

Usage

```
## S4 method for signature 'EMM'
smooth_transitions(x, range = 2, copy = TRUE)
```

Arguments

x	an object of class "EMM"
range	threshold multiplier for the smoothing range.
copy	logical; make a copy of x before reclustering? Otherwise the function will change x!

Value

smooth_transitions returns invisibly an object of class EMM. If copy=FALSE then it returns a reference to the changes object passed as x.

See Also

prune

Examples

```
data("EMMTraffic")

## learn a model
emm <- EMM(threshold=0.2, measure="eJaccard")
build(emm, EMMTraffic)

## smooth the model by adding tansitions
emm_s <- smooth_transitions(emm)

## compare graphs
op <- par(mfrow = c(1, 2), pty = "m")
plot(emm, method="MDS", main="Original")
plot(emm_s, method="MDS", main="Smoothed")
par(op)
```

---

synthetic_stream *Create a Synthetic Data Stream*

Description

This function creates a synthetic data stream with data points in roughly $[0, 1]^p$ by choosing points form k clusters following a sequence through these clusters. Each cluster has a density function following a d-dimensional normal distributions. In the test set outliers are introduced.

Usage

```
synthetic_stream(k = 10, d = 2, n_subseq = 100, p_transition = 0.5, p_swap = 0,
n_train = 5000, n_test = 1000, p_outlier = 0.01, rangeVar = c(0, 0.005))
```

Arguments

k	number of clusters.
d	dimensionality of data set.
n_subseq	length of subsequence which will be repeat to create the data set.
p_transition	probability that the next position in the subsequence will belong to a different cluster.
p_swap	probability that two data points are swapped. This represents measurement errors (e.g., a data points arrive out of order) or that the data stream does not exactly follow the subsequence.
n_train	size of training set (without outliers).
n_test	size of test set (with outliers).

`p_outlier`	probability that a data point is replaced by an outlier (a randomly chosen point in $[0, 1]^p$).
`rangeVar`	Used to create the random covariance matrices for the clusters. See `genPositiveDefMat()` in **clusterGeneration** for details.

Details

The data generation process creates a data set consisting of `k` clusters in roughly $[0, 1]^d$. The data points for each cluster are be drawn from a multivariate normal distribution given a random mean and a random variance/covariance matrix for each cluster. The temporal aspect is modeled by a fixed subsequence (of length `n_subseq`) through the `k` clusters. In each step in the subsequence we have a transition probability `p_transition` that the next data point is in the same cluster or in a randomly chosen other cluster, thus we can create slowly or fast changing data. For the complete sequence, the subsequence is repeated to create `n_test`/`n_train` data points. The data set is generated by drawing a data point from the cluster corresponding to each position in the sequence. Outliers are introduced by replacing data points in the data set with probability `$p_outlier` by randomly chosen data points in $[0, 1]^d$. Finally, to introduce imperfection in the temporal sequence (e.g., because the data does not follow exactly a repeating sequence or because observations do not arrive in the correct order), we swap two consecutive observations with probability `p_swap`.

Value

A list with the following elements:

`test`	test data.
`train`	training data.
`sequence\_test`	sequence of the test data points through the clusters.
`sequence\_train`	sequence of the training data points through the clusters.
`swap\_test`	index where points are swapped.
`swap\_train`	index where points are swapped.
`outlier_position`	logical vector for outliers in test data.
`model`	centers and covariance matrices for the clusters.

Examples

```
## create only test data (with outliers)
ds <- synthetic_stream(n_train=0)

## plot test data
plot(ds$test, pch = ds$sequence_test, col ="gray")
text(ds$model$mu[,1], ds$model$mu[,2], 1:10)

## mark outliers
points(ds$test[ds$outlier_position,], pch=3, lwd=2, col="red")
```

tNN-class *Class "tNN"*

Description

Implements the threshold Nearest Neighbor clustering algorithm used by EMM.

Objects from the Class

Objects can be created with `new()` or by the creator function `tNN`.

Slots

`measure`**:** Object of class `"character"` containing the name of the dissimilarity measure used (see `dist` in **proxy** for available measures)

`centroids`**:** Object of class `"logical"` indicating if centroids are used for clusters. If `FALSE`, pseudo medians (first observation of a cluster) are used to represent a cluster.

`threshold`**:** Object of class `"numeric"` with the dissimilarity threshold used by the NN clustering algorithm for assigning a new observation to existing clusters.

`lambda`**:** Object of class `"numeric"` specifying the rate for fading.

`lambda_factor`**:** Object of class `"numeric"` expressing the fading rate expressed as a factor.

`tnn_d`**:** An environment containing the variable data for the tNN object:

- `centers`**:** Object of class `"matrix"` containing the cluster centers.
- `counts`**:** Object of class `"numeric"` with the number of observations assigned to each cluster.
- `var_thresholds`**:** Object of class `"numeric"` with the dissimilarity thresholds for individual clusters (usually the same as threshold).
- `last`**:** A `"character"` vector containing the cluster names the points for the previous call of `cluster()` were assigned to.

Methods

copy `signature(x = "tNN")`: Make a copy of the tNN object. Making explicit copies is necessary since information is stored in an environment which is not copied for regular assignements.

cluster_centers `signature(x = "tNN")`: returns the cluster centers as a matrix.

cluster_counts `signature(x = "tNN")`: returns the cluster counts as a vector.

clusters `signature(x = "tNN")`: returns the names of the clusters.

last_clustering `signature(x = "tNN")`: returns the indices of the clusters the data points in the last cluster operation where assigned to. To save memory the last clustering information can be removed by setting the formal parameter `remove` to `TRUE`.

nclusters `signature(x = "tNN")`: returns the number of clusters in the clustering.

plot `signature(x = "tNN", y = "missing")`: plots the cluster centers using a scatterplot matrix (see `pairs`).

References

M.H. Dunham, Y. Meng, J. Huang (2004): Extensible Markov Model, In: *ICDM '04: Proceedings of the Fourth IEEE International Conference on Data Mining,* pp. 371–374.

See Also

`cluster` for adding new data to the clustering. `find_clusters` to find the nearest neighbor cluster for given data points. `EMM` extends "tNN".

`TRAC` *TRAC: Creating an EMM from a Regular Clustering*

Description

Create an EMM from a regular clustering (k-means or PAM) of sequence data.

Usage

```
TRAC(x, data = NULL, measure = "euclidean")
```

Arguments

`x`	a clustering object (result of kmeans or pam) or a vector with (integer) cluster assignments.
`data`	the data used for clustering (only used if `x` is a cluster assignment vector).
`measure`	used distance measure.

Details

The order is inferred from the order in the original data set.

Value

A `EMM` object representing the clustering of sequence data.

Examples

```
data("EMMsim")

## using kmeans
cl <- kmeans(EMMsim_train, 10)
emm <- TRAC(cl)
emm
plot(emm, method = "MDS")

## using a cluster assignment vector (taken from the k-means clustering above)
x <- cl$cluster
emm <- TRAC(x, data = EMMsim_train)
```

```
emm
plot(emm, method = "MDS")
```

TRACDS-class *Class "TRACDS"*

Description

Representation of the temporal structure of a data stream clustering using a extensible Markov model.

Objects from the Class

Objects can be created using the creator function `TRACDS` or by directly calling `new("TRACDS", ...)`. Most slots for the extended classes can be used as parameters.

Slots

`lambda`**:** Object of class `"numeric"` specifying the rate for fading.

`lambda_factor`**:** Object of class `"numeric"` expressing the fading rate expressed as a factor.

`tracds_d`**:** An environment containing all the variable data of the TRACDS object:

- `mm`**:** Object of class `"SimpleMC"` representing the first order Markov model of the EMM.
- `current_state`**:** Object of class `"character"` with the name of current state in the EMM. `NA` means no current state.

Methods

copy `signature(x = "TRACDS")`: Make a copy of the TRACDS object. Making explicit copies is necessary since information is stored in an environment which is not copied for regular assignements.

current_state `signature(x = "TRACDS")`: returns the name of the current state.

nstates `signature(x = "TRACDS")`: returns the number of states.

ntransitions `signature(x = "TRACDS")`: returns the number of transitions with a count larger than 0 stored in the object.

plot `signature(x = "TRACDS", y = "missing")`: Plots the object as a directed graph.

states `signature(x = "TRACDS")`: returns the names of the states.

transitions `signature(x = "TRACDS")`: returns all transitions as a matrix of state names with a from and a to column.

Note

A TRACDS object can be coerced to igraph or graph objects using `as.igraph()` and `as.graph()`.

References

M.H. Dunham, Y. Meng, J. Huang (2004): Extensible Markov Model, In: *ICDM '04: Proceedings of the Fourth IEEE International Conference on Data Mining,* pp. 371–374.

M. Hahsler, M. H. Dunham (2010): rEMM: Extensible Markov Model for Data Stream Clustering in R, Journal of Statistical Software, 35(5), 1-31, URL `http://www.jstatsoft.org/v35/i05/`

See Also

Look at `transition`, `transition_matrix` and `initial_transition` to access the transition information in the EMM. `predict` is used to predict future states of an EMM. `EMM` extends `"TRACDS"`.

`transition` *Access Transition Probabilities/Counts in an EMM*

Description

Calculates individual transition probabilities/counts or a complete transition matrix for an EMM (which contains `"TRACDS"`).

Usage

```
## S4 method for signature 'TRACDS,character,character'
transition(x, from, to,
          type = c("probability", "counts", "log_odds"), prior = TRUE)
## S4 method for signature 'TRACDS'
transition_matrix(x,
          type = c("probability", "counts", "log_odds"), prior = TRUE)
## S4 method for signature 'TRACDS'
initial_transition(x,
          type = c("probability", "counts", "log_odds"), prior = TRUE)
```

Arguments

`x`	an object of class `"EMM"`/`"TRACDS"`.
`from, to`	Names a states. If `to` is missing, `from` has to contain a matrix with two columns (a from column and a to column as returned by `transitions`).
`type`	What should be calculated?
`prior`	add one to each transition count. This is equal to starting with a uniform prior for the transition count distribution, i.e., initially all transitions are equally likely.

Details

Log odds are calculated as $ln(a/(1/n))$ where a is the probability of the transition and n is the number of states in the EMM. $1/n$ is the probability of a transition under the null model which assumes that the transition probability from each state to each other state (including staying in the same state) is the same, i.e., the null model has a transition matrix with all entries equal to $1/n$.

Value

A scalar (for `transition`), a square matrix (for `transition_matrix`) or a vector (for `initial_transition`).

See Also

`EMM` which contains `TRACDS`

Examples

```
data("EMMTraffic")
emm <- EMM(measure="eJaccard", threshold=0.2)
emm <- build(emm, EMMTraffic)

## get transition matrix
transition_matrix(emm, type="count", prior=FALSE)
transition_matrix(emm, type="count")
transition_matrix(emm, prior=FALSE)
transition_matrix(emm)

## get initial state probabilities
initial_transition(emm)

## access individual transition probability (state 1 -> 2)
transition(emm, "1","2")

## get counts for all existing transitions
tr <- transitions(emm)
tr
cbind(as.data.frame(tr), counts=transition(emm, tr, type="counts"))
```

`transition_table` *Extract a Transition Table for a New Sequence Given an EMM*

Description

Finds the state sequence of a new sequence in an EMM and returns a table with the transition probabilities or counts.

Usage

```
## S4 method for signature 'EMM,matrix'
transition_table(x, newdata,
  type = c("probability", "counts", "log_odds"),
  match_cluster = "exact", prior=TRUE, initial_transition = FALSE)
```

Arguments

`x`	an `EMM` object.
`newdata`	new sequence,
`type`	the measure to return.
`match_cluster`	do the new observations have to fall within the threshold of the cluster (`"exact"`) or is nearest neighbor used (`"nn"`)?
`prior`	add one to each transition count. This is equal to starting with a uniform prior for the transition count distribution, i.e. initially all transitions are equally likely. It also prevents the product of probabilities to be zero if a transition was never observed.
`initial_transition`	include the initial transition in the table?

Value

A data.frame with three columns (from state, to state and the transition probability/count.)

See Also

`transition` to access transition probabilities and `find_clusters` for assigning observations to states/clusters.

Examples

```
data("EMMsim")

emm <- EMM(threshold=.5)
emm <- build(emm, EMMsim_train)

head(transition_table(emm, EMMsim_test))
head(transition_table(emm, EMMsim_test, type ="prob", initial_transition=TRUE))
```

`update` *Update a TRACDS temporal structure with new state assignements*

Description

Add a sequence of new state assignments to a TRACDS object.

Usage

```
## S4 method for signature 'TRACDS'
update(object, newdata, verbose=FALSE, ...)
reset(x)
compact(x)
```

Arguments

`x,object`	a `TRACDS` object. Note that this function changes the original object!
`newdata`	a vector with a state assignemnt sequence (typically produced by clustering).
`verbose`	logical; verbose output?
`...`	further arguments.

Details

`update()` adds a new state assignemnt sequenc to the TRACDS object by increasing the transition counts and, if needed, creating new states.

`reset()` resets the current state to `NA` for reading in a new sequence. An `NA` in `newdata` also resets the current state.

`compact()` reduces the size (memory) used to store the temporal transition matrix.

Value

A reference to the changed TRACDS object with the data added. Note: EMM objects store all variable data in an environment which enables us to update partial data without copying the whole object. Assignment will not create a copy! Use the provided method `copy()`.

See Also

Class `TRACDS`, `fade`.

Examples

```
## create an empty TRACDS object
tracds <- TRACDS()
tracds

## update with an cluster assignment sequence
update(tracds, c(1,2,5,5,2))
tracds

plot(tracds)
```

Index