Package 'mergeHMM'

June 13, 2013

Type Package

Title Merging states of hidden Markov models.

Version 1.0

Date 2013-05-06

Author Florian Schwaiger

Maintainer Florian Schwaiger <schwaige@mathematik.uni-marburg.de>

Description

In Holzmann and Schwaiger (2013) hidden Markov models with state-dependent finite mixtures are analyzed. The two proposed algorithms to find an appropriate model and to find density based clusters are here implemented for the uni- and multivariate normal distribution.

License GPL-2

Depends methods,sn, mvtnorm, RHmm, parallel, CombMSC

R topics documented:

mergeHMM-package *Investigate the dependence structure of an hidden Markov model*

Description

In Holzmann and Schwaiger (2013) we consider hidden Markov models with state-dependent finite mixtures, here implemented for the univariate and multivariate normal distribution.

This packages provides functions to find an appropriate model by backward selection based on the proposed test (see `findIndependencePartition`). A function to calculate local decoding entropies iteratively, as defined in the paper, is also included (see `investigateMergedHMMs`). This function can be used to find state-dependent distributions which form a density based cluster. Further, functions to simulate datasets, to estimate models via MLE under dependence-structure restrictions or to perform a maximum-a-posteriori analysis with the Viterbi algorithm are included. Finally, to merge states of a given (estimated) HMM we provide the function `mergeHMM`, whereas further explanations of the here considered model class are given in the help file of `mergeHMM`.

Details

Package:	mergeHMM
Type:	Package
Version:	1.0
Date:	2013-05-06
License:	GPL-2

Author(s)

Florian Schwaiger <schwaige@mathematik.uni-marburg.de>

References

Holzmann, H. and Schwaiger, F. (2013). Hidden Markov models with state-dependent mixtures: Minimal representation, model testing and applications to clustering.

Zucchini and MacDonald (2009). Hidden Markov Models for Time Series: An Introduction Using R.

See Also

`findIndependencePartition`, `investigateMergedHMMs`, `simulateHMM`, `mleHMM`, `viterbiHMM` and `mergeHMM`

Examples

```
#setting an HMM which has independence partition {{1,2},{3}}
r1 = c(0.8 * c(0.5,0.5), 0.2)
r3 = c(0.1 * c(0.5,0.5), 0.9)
tpm0 = rbind(r1,r1,r3)
mu0 = c(1,3,7)
```

```
sigma0 = rep(0.8,3)
hmm0= setHMM(transitionMatrix=tpm0,mu=mu0,sigma=sigma0)

#simulating a dataset
set.seed(1)
x = simulateHMM(n=500, hmm=hmm0)$data

#estimate unrestricted HMM
mle0 = mleHMM(x=x,m=3); mle0
#applying Viterbi algorithm
viterbiHMM(x=x,hmm=mle0$mle)

#merging states
mergedModel1 = mergeHMM(hmm=hmm0,G=list(1:2,3)); mergedModel1
mergedModel2 = mergeHMM(hmm=mle0$mle,G=list(1:2,3)); mergedModel2

#for examples of the main algorithms see the help files of
#findIndependencePartition and investigateMergedHMMs
```

`findIndependencePartition` *Find independence partition.*

Description

This function finds via backward selection the independence partition of the HMM given a dataset, see Holzmann and Schwaiger (2013).

Usage

```
findIndependencePartition(m, x, alpha = 0.05, estimateIn, details = FALSE, hmm =
```

Arguments

`m`	the number of states of the initial HMM
`x`	the dataset
`alpha`	the level for iterative testing, we recommend 0.01
`estimateIn`	if the initial model is already estimated (which can be time consuming) it can be supplied to the algorithm here
`details`	TRUE/FALSE: controls the output object (how many details)
`hmm`	if estimateIn is not supplied: optionally an object of type `hmmClass-class` can be supplied as staring point for iterative optimization of the initial model, otherwise a (unrestricted) starting HMM is calculated using the package RHmm
`...`	additional optional parameters passed to `mleHMM`

Value

a list

`pValuePath`	the p-values
`finalModel`	the final model

`finalG` the according final partition -> the found independence partition

only if details=TRUE:

`modelPath` a list containing the p-value optimal model in each step of the iterative algorithm

`GPath` a list containing the partitions according to the models in the list modelPath

`loglikePath` the values of the log-likelihoods

`likelihoodRatioPath`
the values of the likelihood ratios

References

Holzmann, H. and Schwaiger, F. (2013). Hidden Markov models with state-dependent mixtures: Minimal representation, model testing and applications to clustering.

See Also

`investigateMergedHMMs`

Examples

```
#setting an HMM which has independence partition {{1,2,3},{4}}
r1 = c(0.8 * c(1/3,1/3,1/3), 0.2)
r4 = c(0.1 * c(1/3,1/3,1/3), 0.9)
tpm0 = rbind(r1,r1,r1,r4)
mu0 = c(1,4,9,15)
sigma0 = rep(0.8,4)
hmm0= setHMM(transitionMatrix=tpm0,mu=mu0,sigma=sigma0)

#simulating a dataset
set.seed(1)
x = simulateHMM(n=500, hmm=hmm0)$data
plot(density(x))
rug(x)

#find at first independence partition and save it
indepObj= findIndependencePartition(m=4,x=x,alpha=0.01)

estHmm = indepObj$finalModel
estPartition = indepObj$finalG

#find density based cluster using LDE and dependence structure restriction
obj0  = investigateMergedHMMs(x=x,hmm=estHmm,independenceClusters=estPartition)

#seek for an elbow -> second model
plot(obj0$localDecodingEntropies,type="b",pch=19)

finalHmm = obj0$modelPath[[2]] ; finalHmm
#third state could be included (w.r.t. dependence structure),
#is not since it would not produce a density based cluster,
#LDE finds here the right model...
```

`hmmClass-class` *Class for hiddenMarkovModels.*

Description

Hidden Markov model class.

Objects from the Class

Objects should be set with the function `setHMM`.

Slots

Each model has the following five slots.

`distribution`**:** a character, specifies the state-dependent distribution.

`transitionMatrix`**:** a square matrix, represents the transition probability matrix

`stateParameters`**:** a list, specifies the state-dependent parameters of the HMM

`probList`**:** a list, specifies the weights of the state-dependent finite mixtures.

`internVariables`**:** a list, contains several intern variables.

Methods

For each model the following generic functions are provided:

investigateMergedHMMs: find density based cluster

mergeHMM: merge states of the HMM

reduceHMM: reduce dependence information of the HMM

simulateHMM: simulate a dataset

viterbiHMM: apply Viterbi algorithm for maximum a-posteriori estimation

Extends

The classes

`hmmClassNormal`

`hmmClassMvNormal`

extend the class `hmmClass`.

See Also

`setHMM`

`investigateMergedHMMs` *Find density based clusters*

Description

This function implements the clustering algorithm using the local decoding entropy (LDE) as proposed in Holzmann and Schwaiger (2013). The HMM and its independence clusters have to be estimated before. To find the independence partition and estimate the model use the function `findIndependencePartition`. In order to find the final model plot the values of the LDE, see the example below.

Usage

```
investigateMergedHMMs(x, hmm, independenceClusters)
```

Arguments

`x`	dataset
`hmm`	an object of type `hmmClass-class`, no state-dependent mixtures, since it is the initial model. Typically estimated using the function `findIndependencePartition`.
`independenceClusters`	a list, which is a partition of the state space 1,...,k (each list entry is thus a vector), this partition defines the flexibility for merging. Also obtained from `findIndependencePartition`, see the example below.

Value

a list

`modelPath`	a list containing the local decoding optimal model in each step of the iterative algorithm
`GPath`	a list containing the partitions according to the models in the list modelPath
`localDecodingEntropies`	the values of the local decoding entropies

References

Holzmann, H. and Schwaiger, F. (2013). Hidden Markov models with state-dependent mixtures: Minimal representation, model testing and applications to clustering.

See Also

`findIndependencePartition`

Examples

```
#setting an HMM which has independence partition {{1,2,3},{4}}
r1 = c(0.8 * c(1/3,1/3,1/3), 0.2)
r4 = c(0.1 * c(1/3,1/3,1/3), 0.9)
tpm0 = rbind(r1,r1,r1,r4)
mu0 = c(1,4,9,15)
sigma0 = rep(0.8,4)
hmm0= setHMM(transitionMatrix=tpm0,mu=mu0,sigma=sigma0)

#simulating a dataset
set.seed(1)
x = simulateHMM(n=500, hmm=hmm0)$data
plot(density(x))
rug(x)

#find at first independence partition and save it
indepObj= findIndependencePartition(m=4,x=x,alpha=0.01)

estHmm = indepObj$finalModel
estPartition = indepObj$finalG

#find density based cluster using LDE and dependence structure restriction
obj0  = investigateMergedHMMs(x=x,hmm=estHmm,independenceClusters=estPartition)

#seek for an elbow -> second model
plot(obj0$localDecodingEntropies,type="b",pch=19)

finalHmm = obj0$modelPath[[2]] ; finalHmm
#third state could be included (w.r.t. dependence structure),
#is not since it would not produce a density based cluster,
#LDE finds here the right model...
```

---

localDecodingEntopie

Calculate Local decoding entropies.

Description

The local decoding entropy (LDE) is the sum of the entropies of the a-posteriori probabilities for the hidden Markov chain to attain at time t state j (j=1,..,k) given the whole observable process, see Holzmann an Schwaiger (2013). This function calculates the LDE of an HMM when merging the states w.r.t. partition G. The input has to be an usual HMM (non-mixture state-dependent distributions) and the partition G which gives the states to be merged. The reason for this parametrization is, that so we can easily compare LDE-values when merging w.r.t. different partitions. We adapted the functions of Zucchini (2009) for calculating local decoding probabilities to the setting with potentially mixtures as state dependent distributions.

Usage

```
localDecodingEntopie(x, hmm, G, details = FALSE)
```

Arguments

x	dataset
hmm	an object of type hmmClass-class, no state-dependent mixtures, use G for that.
G	a list, which is a partition of the state space 1,...,k (each list entry is thus a vector), see the example below.
details	if TRUE, also the a-posteriori probabilities are returned

Value

a list with entries depending on variable details

entropy	the local decoding entropy
probs	the local decoding a-posteriori probabilities

References

Holzmann, H. and Schwaiger, F. (2013). Hidden Markov models with state-dependent mixtures: Minimal representation, model testing and applications to clustering.

Zucchini and MacDonald (2009). Hidden Markov Models for Time Series: An Introduction Using R.

Examples

```
#setting an HMM which has independence partition {{1,2},{3}}
r1 = c(0.8 * c(0.5,0.5), 0.2)
r3 = c(0.1 * c(0.5,0.5), 0.9)
tpm0 = rbind(r1,r1,r3)
mu0 = c(1,3,7)
sigma0 = rep(0.8,3)
hmm0= setHMM(transitionMatrix=tpm0,mu=mu0,sigma=sigma0)

set.seed(1)
x = simulateHMM(n=500, hmm=hmm0)$data

#LDE of HMM under under different restrictions
localDecodingEntopie(x=x,hmm=hmm0,G=as.list(1:3))
localDecodingEntopie(x=x,hmm=hmm0,G=list(1:2,3))
localDecodingEntopie(x=x,hmm=hmm0,G=list(1:3))
```

---

mergeHMM *Merging a hidden Markov model.*

Description

This page documents the function mergeHMM. For the package overview see mergeHMM-package.

The function mergeHMM merges states of an HMM given a partition of the state space 1,...,k of the according hidden Markov chain, i.e. an HMM is mapped onto a new HMM. Note that the given partition should divide the state space into r<k groups. The new HMM has a transition

probability matrix with entries being the transition probabilities between the groups of the given partition w.r.t. the input transition probability matrix. The new state dependent distributions are given by mixtures of the input state dependent distributions, whereas the mixture proportions are given by the (stationary) conditional state probabilities within the according group, see the reference and the example below.

Usage

```
mergeHMM(hmm, G)
```

Arguments

hmm	an object of type hmmClass-class. For creating such an object use setHMM or functions which estimate it, i.e. findIndependencePartition.
G	a list, which is a partition of the state space 1,...,k (each list entry is thus a vector), see the example below.

Details

Consider the input HMM to have k states. Than the partition G has to be a list with r<k entries, whereas each list entry (an integer vector) represents a group of states being a subset of the state space 1,...,k. All list entries have to be disjoint. The output HMM has r states. The transition probability matrix is given by the group transition probabilities induced by the input transition matrix and the groups given by G. As described above the state-dependent distributions of the merged model are mixtures given by the (stationary) state probabilities within the groups.

The observable process of the resulting HMM has exactly the same distribution as the observable part of a reduced HMM, which can be found using the function reduceHMM. The difference between both models is only the state interpretation: In the reduced HMM we still deal with k states. In case of merging the latent mixture states are not interpreted as states anymore and the mapping results in an r state HMM.

Note, that both functions in general change the time-series model. Only iff the transition matrix has a special structure the probabilistic structure of the observable process in both cases is not changed. To check whether a transition matrix possesses has this structure, it simply has to be checked if the matrix does not change when the function reduceTpm is applied. For more theoretical background see the paper below.

Value

an object of type hmmClass-class.

Note

The merged HMM is here NOT estimated in the restricted parameter space. No dataset is needed, it is simply a mapping of the model. For estimating under the restriction that state dependent distributions are mixtures, use the function mleHMM and see the the examples there.

References

Holzmann, H. and Schwaiger, F. (2013). Hidden Markov models with state-dependent mixtures: Minimal representation, model testing and applications to clustering.

See Also

mergeHMM-package, mleHMM, reduceHMM, reduceTpm and setHMM.

Examples

```
#first example:
#  setting a five state HMM with strongly persistent structure in the hidden Markov chain
#  the dependence structure will change under merging or reducing

#settinng the input HMM (five states)
tpm1 = matrix(c(0.92,0.02,0.02,0.02,0.02,
                0.02,0.92,0.02,0.02,0.02,
                0.02,0.02,0.92,0.02,0.02,
                0.02,0.02,0.02,0.92,0.02,
                0.02,0.02,0.02,0.02,0.92),ncol=5,nrow=5,byrow=TRUE)
mu0 = c(1,3,6,10,15)
sigma0 = rep(1,5)
hmmIn = setHMM(transitionMatrix=tpm1,mu=mu0,sigma=sigma0)

#setting a partition
# -> sets states 1 and 2, 3 and 4 and 5 each to one group
G_in = list(1:2,3:4,5)

mergeHMM(hmm=hmmIn,G=G_in)
reduceHMM(hmm=hmmIn,G=G_in) # tpm does change in comparison to tpm1!

#second example:
#  setting a five state HMM with potential for merging
#  the dependence structure of the model will not change under merging or reducing
r12 = c( 0.7 * c(0.9,0.1), 0.2  * c(0.2,0.8), 0.1  )
r34 = c( 0.1 * c(0.9,0.1), 0.8  * c(0.2,0.8), 0.1  )
r5   = c( 0.1 * c(0.9,0.1), 0.05 * c(0.2,0.8), 0.85  )

tpm2 = rbind(r12,r12,r34,r34,r5)
hmmIn2 = setHMM(transitionMatrix=tpm2,mu=mu0,sigma=sigma0)


mergeHMM(hmm=hmmIn2,G=G_in)
reduceHMM(hmm=hmmIn2,G=G_in) # tpm does NOT change in comparison to tpm2!
```

---

mleHMM *Estimating (restricted) HMMs.*

Description

This function estimates the (restricted) MLE given a dataset. If restrictions to the dependence structure should be applied, this can be done via the argument G, see the details section and the examples.

Usage

```
mleHMM(x, m, G = as.list(1:m), hmm = NULL, ...)
```

Arguments

x	the dataset. If it is a vector the univariate normal distribution is used, if it is a matrix (dimensions row-wise) the multivariate normal distribution is used.
m	number of states of the Markov chain
G	optionally a list, which is a partition of the state space 1,...,k (each list entry is thus a vector). This implies a restriction on the dependence structure, see below.
hmm	optionally an object of type hmmClass-class can be supplied as staring point for iterative optimization, otherwise a (unrestricted) starting HMM is calculated using the package RHmm
...	If no starting parameter for optimization is supplied (hmm=NULL), then the starting parameter is calculated by using the function HMMfit of the package RHmm by fitting an unristricted HMM. Thus this argument can be used to pass additional control parameters to the function HMMFit, see the help file for further details on that. If a starting HMM is supplied, then this argument can be used to pass additional parameter to nlm, which calculates iteratively the restricted MLE, see the help file of nlm for further details.

Details

With the argument G restrictions to the transition matrix can be applied. Using G = as.list(1:m) gives an unrestricted usual HMM, using G = list(1:m) gives an i.i.d. model. All cases in between are also possible, see the example below. When using restricted estimation a starting value should be supplied because of possible label switching, e.g. by unrestricted estimation before (see example below).

Value

A list with entries

mle	object of type hmmClass-class
code	convergence-flag of nlm
mllk	value of optimal minus log-likelihood
AIC	AIC
BIC	BIC

See Also

nlm and HMMFit

Examples

```
#setting an HMM which has independence partition {{1,2},{3}}
r1 = c(0.8 * c(0.5,0.5), 0.2)
r3 = c(0.1 * c(0.5,0.5), 0.9)
tpm0 = rbind(r1,r1,r3)
mu0 = c(1,3,7)
sigma0 = rep(0.8,3)
hmm0= setHMM(transitionMatrix=tpm0,mu=mu0,sigma=sigma0)

#simulating a dataset
```

```
set.seed(1)
x = simulateHMM(n=500, hmm=hmm0)$data

#estimate unrestricted HMM
mle0 = mleHMM(x=x,m=3); mle0

#estimate with correct restriction
mle1 = mleHMM(x=x,m=3,G=list(1:2,3),hmm=mle0$mle); mle1

#estimate with wrong restriction
mle2 = mleHMM(x=x,m=3,G=list(1,2:3),hmm=mle0$mle); mle2
```

`reduceHMM` *Reducing a hidden Markov model.*

Description

The function `reduceHMM` reduces the dependence structure of the hidden Markov chain of an HMM given a partition of the state space 1,...,k, i.e. an HMM is mapped onto a new HMM. Note that the given partition should divide the state space into r<k groups. The new transition probability from state i to j results by splitting the original state migration into two steps: It is the product of the probability from switching from the group where i is in to the group where j is in AND the probability for state j given the Markov chain is in the group where j is in. The function is detailedly defined in Holzmann and Schwaiger (2013).

The distribution of the observable part (as hole process) of the reduced HMM is the same as for the merged HMM, see `mergeHMM`. In general both models differ from the original one.

Usage

```
reduceHMM(hmm, G)
```

Arguments

`hmm`	an object of type `hmmClass-class`. For creating such an object use `setHMM`.
`G`	a list, which is a partition of the state space 1,...,k (each list entry is thus a vector), see the example below.

Value

an object of type `hmmClass-class`.

References

Holzmann, H. and Schwaiger, F. (2013). Hidden Markov models with state-dependent mixtures: Minimal representation, model testing and applications to clustering.

See Also

`mergeHMM`

Examples

```
#first example:
#  setting a five state HMM with strongly persistent structure in the hidden Markov chain
#  the dependence structure will change under merging or reducing

#settinng the input HMM (five states)
tpm1 = matrix(c(0.92,0.02,0.02,0.02,0.02,
                0.02,0.92,0.02,0.02,0.02,
                0.02,0.02,0.92,0.02,0.02,
                0.02,0.02,0.02,0.92,0.02,
                0.02,0.02,0.02,0.02,0.92),ncol=5,nrow=5,byrow=TRUE)
mu0 = c(1,3,6,10,15)
sigma0 = rep(1,5)
hmmIn = setHMM(transitionMatrix=tpm1,mu=mu0,sigma=sigma0)

#setting a partition
# -> sets states 1 and 2, 3 and 4 and 5 each to one group
G_in = list(1:2,3:4,5)

mergeHMM(hmm=hmmIn,G=G_in)
reduceHMM(hmm=hmmIn,G=G_in) # tpm does change in comparison to tpm1!

#second example:
#  setting a five state HMM with potential for merging
#  the dependence structure of the model will not change under merging or reducing
r12 = c( 0.7 * c(0.9,0.1), 0.2  * c(0.2,0.8), 0.1  )
r34 = c( 0.1 * c(0.9,0.1), 0.8  * c(0.2,0.8), 0.1  )
r5   = c( 0.1 * c(0.9,0.1), 0.05 * c(0.2,0.8), 0.85  )

tpm2 = rbind(r12,r12,r34,r34,r5)
hmmIn2 = setHMM(transitionMatrix=tpm2,mu=mu0,sigma=sigma0)


mergeHMM(hmm=hmmIn2,G=G_in)
reduceHMM(hmm=hmmIn2,G=G_in) # tpm does NOT change in comparison to tpm2!
```

`reduceTpm` *Reducing dependence information of a Markov chain.*

Description

This is the same transformation as in `reduceHMM` for the hidden Markov chain, but here the input/output is only a transition probability matrix.

Usage

```
reduceTpm(transitionMatrix, G)
```

Arguments

`transitionMatrix`	a transition probability matrix of dimension (k x k).
`G`	a list, which is a partition of the state space 1,...,k (each list entry is thus a vector), see the example below.

Value

a transition probability matrix of dimension (k x k).

See Also

reduceHMM

Examples

```
#first example
tpm1 = matrix(c(0.92,0.02,0.02,0.02,0.02,
                0.02,0.92,0.02,0.02,0.02,
                0.02,0.02,0.92,0.02,0.02,
                0.02,0.02,0.02,0.92,0.02,
                0.02,0.02,0.02,0.02,0.92),ncol=5,nrow=5,byrow=TRUE)
G_in = list(1:2,3:4,5)
reduceTpm(transitionMatrix=tpm1,G=G_in)

#second example
r12 = c( 0.7 * c(0.9,0.1), 0.2  * c(0.2,0.8), 0.1  )
r34 = c( 0.1 * c(0.9,0.1), 0.8  * c(0.2,0.8), 0.1  )
r5   = c( 0.1 * c(0.9,0.1), 0.05 * c(0.2,0.8), 0.85  )
tpm2 = rbind(r12,r12,r34,r34,r5)
reduceTpm(transitionMatrix=tpm2,G=G_in)
```

---

setHMM *Setting a hidden Markov model.*

Description

This function should be used in order to set an object of type hmmClass-class. Models with (mixtures of) normal or multivariate normal state-dependent distributions are implemented. For the exact parametrization see the details section.

The functions mleHMM and findIndependencePartition estimate HMMs and use the class hmmClass-class as output.

Usage

```
setHMM(transitionMatrix, mu, sigma, probList = as.list(rep(x = 1, times = dim(tr
```

Arguments

transitionMatrix	a square matrix, which is a transition matrix
mu	the mean parameters, vector or matrix, see details
sigma	the scale parameters, vector or array, see details
probList	the probabilities of state-dependent finite mixtures, see details

Details

The parametrization is as follows: We consider the situation where the number of all mixture states equals k. Thus there are k mean parameters and k variance parameters. In the univariate case we thus have two vectors of dimension k. In the multivariate case (with dimension = d) each mean parameter has d entries, thus we parametrize the hole mean parameter of the HMM by a (d x k) matrix, i.e. the dimensions are row-wise. For the variance parameter there is for each state one (d x d) matrix, therefore the variance parameter of the whole model is parametrized by an array of dimension (d x d x k).

In order to allow for mixtures there is the argument `probList`. As default it is a list with k entries where each entry equals one. This parametrizes a common HMM with k states and normal state dependent distributions.

If e.g. the first state dependent distribution should be a mixture with k1 components then the first list element of `probList` has to be a k1-dimensional probability vector (positive and summing up to one). Further the first k1 elements in the mean and variance parameter correspond to this mixture. For the second and further states one proceeds accordingly.

Note, that the number of list entries of `probList` (e.g. r) has to be suitable to the given selected transition probability matrix, i.e. the dimension has to be (r x r). In case of some distribution are mixtures and some are just normals all list entries in `probList` have to be entered (although some are one). In case of no mixtures r simply equals k.

Value

an object of type `hmmClass-class`.

Examples

```
#example 1: setting a usual (univariate) normal HMM
tpm1 = matrix(c(0.92,0.02,0.02,0.02,0.02,
                0.02,0.92,0.02,0.02,0.02,
                0.02,0.02,0.92,0.02,0.02,
                0.02,0.02,0.02,0.92,0.02,
                0.02,0.02,0.02,0.02,0.92),ncol=5,nrow=5,byrow=TRUE)
mu1 = c(1,3,6,10,15)
sigma1 = rep(1,5)
hmm1 = setHMM(transitionMatrix=tpm1,mu=mu1,sigma=sigma1); hmm1

#example 2: setting a two state HMM with state-dependent mixtures of (univariate) normals
tpm2 = matrix(c(0.9,0.1,
   0.1,0.9),ncol=2,nrow=2,byrow=TRUE)
hmm2 = setHMM(transitionMatrix=tpm2,mu=mu1,sigma=sigma1,probList=list(c(1/3,1/3,1/3),c(0.

#example3: setting a two state HMM with state-dependent mixtures of (bivariate) normals

sigma0 = array(dim=c(2,2,5))
sigma0[,,1] = matrix(0.3*c(1,0.6,0.6,1),byrow=TRUE,ncol=2)
sigma0[,,2] = matrix(0.3*c(1,-0.6,-0.6,1),byrow=TRUE,ncol=2)
sigma0[,,3] = matrix(1.2*c(0.4,-0.35,-0.35,0.4),byrow=TRUE,ncol=2)
sigma0[,,4] = matrix(0.3*c(4,0.9,0.9,4),byrow=TRUE,ncol=2)
sigma0[,,5] = matrix(0.5*c(1,0.8,0.8,1) ,byrow=TRUE,ncol=2)


mu0 = NA * matrix(0,ncol=5,nrow=2)
mu0[,1] = c(2.5,1.5)
```

```
mu0[,2] = c(3.5,2)
mu0[,3] = c(2,7)
mu0[,4] = c(3,0.5)
mu0[,5] = c(2.5,6)

hmm3 = setHMM(transitionMatrix=tpm2,mu=mu0,sigma=sigma0,probList=list(c(0.2,0.5,0.15,0.15
```

`show-methods` *Output of HMMs on console.*

Description

Provides for each HMM an output on the console.

`simulateHMM` *Simulate a dataset.*

Description

This function simulates a dataset of a given HMM.

Usage

```
simulateHMM(n, hmm)
```

Arguments

`n`	the sample size
`hmm`	the HMM, i.e. an object of type `hmmClass-class`

Examples

```
#setting an HMM which has independence partition {{1,2},{3}}
r1 = c(0.8 * c(0.5,0.5), 0.2)
r3 = c(0.1 * c(0.5,0.5), 0.9)
tpm0 = rbind(r1,r1,r3)
mu0 = c(1,3,7)
sigma0 = rep(0.8,3)
hmm0= setHMM(transitionMatrix=tpm0,mu=mu0,sigma=sigma0)

#simulating a dataset
set.seed(1)
x = simulateHMM(n=500, hmm=hmm0)$data
plot(density(x))
```

stat_distr *Calculating the stationary distribution of a MC*

Description

This function can be used to calculate the stationary distribution of a given Markov chain (parameterized by a transition probability matrix `mat`). This function is not written by us (see reference), but very useful when working with the present type of models, and therefore added to the package.

Usage

```
stat_distr(mat)
```

Arguments

`mat`	The tpm of a Markov chain.

Value

A vector of probabilities, which is the stationary distribution of the Markov chain.

Author(s)

Zucchini and MacDonald (2009)

References

Zucchini and MacDonald (2009). Hidden Markov Models for Time Series: An Introduction Using R.

viterbiHMM *Viterbi algorithm for normal-mixture HMMs.*

Description

Given a dataset and an object of type `hmmClass-class` this function calculates the most likely sequence of states. The state dependent distributions can be mixtures of normal or just normal distributions.

Usage

```
viterbiHMM(x, hmm)
```

Arguments

`x`	a dataset. For the univariate case a vector for the multivariate case a matrix, where the dimensions have to be row-wise.
`hmm`	an object of type `hmmClass-class`. For creating such an object use `setHMM`.

Details

The code in Zucchini and MacDonald (2009) was adopted by us to the setting with potentially mixtures as state dependent distributions.

Value

a vector with the most likely sequence of states.

References

Zucchini and MacDonald (2009). Hidden Markov Models for Time Series: An Introduction Using R.

Examples

```
#set HMM
tpm0 = matrix(c(0.8,0.2,0.1,0.9),ncol=2,nrow=2,byrow=TRUE)
pList0 = list(c(0.4,0.3,0.3),c(1))
mu0 = c(-4,0,2,10)
sigma0 = c(0.3,0.3,0.3,3)
hmm1 = setHMM(transitionMatrix=tpm0,mu=mu0,sigma=sigma0,probList=pList0)

#simulate data
dataset = simulateHMM(n=1000,hmm=hmm1)
x = dataset$data
plot(density(x))

#apply viterbi algorithm
viterbiHMM(x=x,hmm=hmm1)
```

Index