Package 'tileHMM'

June 3, 2008

Type Package

Title Hidden Markov Models for ChIP-on-Chip Analysis

Version 1.0-1

Date 2008-02-27

Author Peter Humburg

Maintainer Peter Humburg <Peter.Humburg@csiro.au>

Description This package provides methods and classes to build HMMs that are suitable for the analysis of ChIP-on-chip data. The provided parameter estimation methods include the Baum-Welch algorithm and Viterbi training as well as a combination of both.

Depends R (>= 2.4.0)

Imports methods

Suggests st, affy, geneplotter

License GPL version 2 or newer

R topics documented:

`baumWelch` *Baum-Welch Algorithm*

Description

The Baum-Welch algorithm [Baum et al., 1970] is a well established method for estimating parameters of HMMs. It represents the EM algorithm [Dempster et al., 1977] for the specific case of HMMs. The formulation of the Baum-Welch algorithm used in this implementation is based on the description given by Rabiner [1989].

Usage

```
## S4 method for signature 'hmm, list':
baumWelch(hmm, obs, max.iter=FALSE, eps=0.01, df=NULL,
        trans.prior=NULL, init.prior=NULL, verbose=1)
```

Arguments

`hmm`	An object of class `hmm` or one of its subclasses representing the hidden Markov model.
`obs`	A list of observation sequences.
`max.iter`	Maximum number of iterations. (optional)
`eps`	Minimum difference in log likelihood between iterations. Default: 0.01
`df`	If this is `NULL` the degrees of freedom for the t distributions are estimated from the data. Otherwise they are set to `df`.
`trans.prior`	Prior distribution of transition probabilities. A prior can be specified either by providing a matrix with transition probabilities or by setting `trans.prior=TRUE`. In the latter case the initial parameter estimates are used as prior. If `trans.prior` is `NULL` (the default) no prior is used.
`init.prior`	Prior distribution of initial state probabilities. A prior can be specified either by providing a vector with initial state probabilities or by setting `init.prior=TRUE`. In the latter case the initial parameter estimates are used as prior. If `init.prior` is `NULL` (the default) no prior is used.
`verbose`	Level of verbosity. Allows some control over the amount of output printed to the console.

Value

Returns the HMM with optimised parameters.

Author(s)

Peter Humburg

References

Baum, L. E. and Petrie, T. and Soules, G. and Weiss, N. 1970 A maximization technique occuring in the statistical analysis of probabilistic functions of markov chains. *The Annals of Mathematical Statistics*, **41**(1), 164–171.

Dempster, A. P. and Laird, N. M. and Rubin, D. B. 1977 Maximum likelihood for incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, Series B, **39**(1).

Rabiner, L. R. 1989 A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, **77**(2), 257–286.

See Also

`viterbiTraining`, `viterbiEM`, `getHMM`, `hmm.setup`

Examples

```
## create two state HMM with t distributions
state.names <- c("one","two")
transition <- c(0.035, 0.01)
location <- c(1, 2)
scale <- c(1, 1)
df <- c(4, 6)
hmm1 <- getHMM(list(a=transition, mu=location, sigma=scale, nu=df),
    state.names)

## generate observation sequences from model
obs.lst <- list()
for(i in 1:50) obs.lst[[i]] <- sampleSeq(hmm1, 100)

## fit an HMM to the data (with fixed degrees of freedom)
hmm2 <- hmm.setup(obs.lst, state=c("one","two"), df=5)
hmm2.fit <- baumWelch(hmm2, obs.lst, max.iter=20, df=5, verbose=1)

## fit an HMM to the data, this time estimating the degrees of freedom
hmm3.fit <- baumWelch(hmm2, obs.lst, max.iter=20, verbose=1)
```

---

contDist-class *Class "contDist"*

Description

Class for continuous probability distributions.

Objects from the Class

Objects can be created by calls of the form `new("contDist", weight, center, disp)`. These objects represent a continuous probability distribution as a mixture of one or more elementary distributions.

Slots

components: Object of class `"matrix"` with one row for each mixture component and a column for each of mixture weight, location parameter and dispersion parameter.

Extends

Class `"dist"`, directly.

Methods

No methods defined with class "dist" in the signature.

Author(s)

Peter Humburg

See Also

`tDist`, `discDist`

Examples

```
showClass("contDist")
```

---

contHMM-access *Accessing Objects of Class "contHMM"*

Description

Access to model parameters and densities of emission distributions.

Usage

```
## S3 method for class 'contHMM':
x[i, j, transition = TRUE, log = FALSE, sum = TRUE, ...]
```

Arguments

`x`	Object of class `contHMM`
`i`	State for which parameter values should be retrieved. This can either a numeric value giving the state index or a character string with the state name.
`j`	Second index identifying parameter (see Details).
`transition`	Logical indicating whether transition probabilities or density values should be returned.

log	Logical indicating whether values should be log transformed before they are returned.
sum	Logical indicating whether densities of mixture components should be summed up. This is ignored if transition = TRUE.
...	Futher arguments to be passed to and from other methods.

Details

This function allows access to the transition probability matrix of the model as well as the emission distributions. If transition = TRUE the transition probability matrix is accessed. In this case i and j identify rows and columns of the matrix respectively. Both can be given as either numeric index or name of the respective states. Either or both of i and j may be missing to indicate that an entire row or column should be selected.

If transition = FALSE the emission distribution of state i is accessed instead. In this case the density function is evaluated at point j.

Value

Either a subset of the transition probability matrix of x or the probability density of state i evaluated at point j (see Details).

Author(s)

Peter Humburg

See Also

contHMM

Examples

```
## create two state HMM with t distributions
state.names <- c("one","two")
transition <- c(0.1, 0.02)
location <- c(1, 2)
scale <- c(1, 1)
df <- c(4, 6)
hmm <- getHMM(list(a=transition, mu=location, sigma=scale, nu=df),
    state.names)

## transition probability from state 'one' to state 'two'
hmm["one", "two"]
## or equivalently
hmm[1, 2]

## get the transition probability matrix
hmm[ , ]

## evaluate emission distribution function of state 'one' at 0
hmm["one", 0, transition = FALSE]

## again, this time using log transformation
hmm["one", 0, transition = FALSE, log = TRUE]
```

`contHMM-class` *Class "contHMM"*

Description

Class for HMMs with continuous emission distributions.

Objects from the Class

Objects can be created by calls of the form `new("contHMM", transition, emission, init)`. For the special case of *t* distributions objects of this class can be created more conveniently by a call to `getHMM`. The function `hmm.setup` provides facilities to create `"contHMM"` objects with initial parameter estimates obtained from data.

Slots

`transition.matrix:` Object of class `"matrix"`, storing the transition probabilities of the Markov chain.

`emission:` Object of class `"list"` containing objects of class `"contDist"` to represent emission distributions for each state.

`init:` Object of class `"numeric"`. The initial state distribution of the Markov chain.

Extends

Class `"hmm"`, directly.

Methods

sampleSeq `signature(hmm = "contHMM", size = "numeric")`: Sample observation sequence of length `size` from model

Author(s)

Peter Humburg

See Also

`contDist`, `tDist`,`hmm`, `sampleSeq`, `baumWelch`, `viterbiTraining`, `viterbi`, `forward`, `backward`, `states`

Examples

```
showClass("contHMM")
```

discDist-class *Class "discDist"*

Description

Class for discrete probability distributions.

Objects from the Class

Objects can be created by calls of the form `new("discDist", alpha, prob, dstr)`. Either `alpha` and `prob` or `dstr` have to be provided. The resulting object represents a discrete probability distribution over alphabet `alpha` where the probabilities for individual symbols are given by `prob`.

Slots

`alpha`: A character vector containing all symbols of the alphabet.

`prob`: Numeric vector of same length as `alpha` with probabilities for each symbol.

`dstr`: A `list` of the form `symbol = probability`.

Extends

Class `"dist"`, directly.

Methods

sampleObs `signature(dist = "discDist", size = "numeric")`: sample observations from `alpha` with probability `prob`.

Author(s)

Peter Humburg

See Also

`contDist`, `tDist`

Examples

```
showClass("discDist")
```

dist-access *Accessing and Converting Objects of Class "dist"*

Description

These methods provide convenient access to objects of class `dist` as well as conversion to other data structures.

Usage

```
## S3 method for class 'discDist':
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
## S3 method for class 'discDist':
as.vector(x, mode = "any")
## S3 method for class 'discDist':
as.matrix(x, ...)

## S4 method for signature 'discDist':
length(x)

## S3 method for class 'discDist':
x[i, ...]
## S3 method for class 'discDist':
x[[i, ...]]
## S3 method for class 'contDist':
x[i, j, ...]
## S3 method for class 'tDist':
x[i, j, log=FALSE, ...]

## S3 replacement method for class 'contDist':
x[i, j] <- value
## S3 replacement method for class 'tDist':
x[i, j] <- value
```

Arguments

`x`	Object of class `discDist`, `contDist` or `tDist`.
`i`	If `x` is an object of class `discDist` this is expected to be a symbol from the alphabet of `x`. For all other classes `i` is interpreted as the index of the mixture component that should be accessed. `i` may be missing in which case values for all mixture components are returned or replaced.
`j`	Either a character string identifying one of the parameters of `x` or a numeric value. In the later case the density at point `j` is returned.
`log`	Logical indicating whether the density value should be log transformed.
`row.names`	NULL or a character vector giving the row names for the data frame. Missing values are not allowed.
`optional`	logical. If TRUE, setting row names and converting column names (to syntactic names) is optional.
`mode`	Mode of `vector`.

value	New parameter value(s).
...	Additional arguments.

Value

The conversion functions return objects of the respective class.

length returns the number of symbols in the alphabet of x.

The access functions return either the requested parameter value or the value of the dansity function at point j.

Author(s)

Peter Humburg

See Also

dist

Examples

```
## converting "discDist" objects
nucleo <- new("discDist",alpha=c("A","C","G","T"),prob=rep(0.25,times=4))
nucleo.vec <- as.vector(nucleo)
nucleo.mat <- as.matrix(nucleo)
nucleo.df <- as.data.frame(nucleo)

## get number of symbols in alphabet
len <- length(nucleo)   # = 4

## get probability for symbol 'A'
nucleo['A']  # = 0.25

## accessing tDist objects
mydt <- new("tDist", mean=0, var=1, df=3)

## evaluate density function
mydt[, 2]  # = 0.06750966
mydt[, 1000] # = 3.307954e-12
mydt[, 1000, log=TRUE]  # = -26.43469

## access parameter values
mydt[, "mean"]   # = 0
```

---

dist-class *Class "dist"*

Description

Class to represent distribution functions.

Objects from the Class

This is a virtual class. Create objects of drived classes instead.

Methods

No methods defined with class "dist" in the signature.

Author(s)

Peter Humburg

See Also

discDist, contDist, tDist

Examples

```
showClass("dist")
```

---

forward *Computation of Forward and Backward Variables*

Description

These functions calculate the forward and backward variables for a given model and observation sequence. All computations are carried out in log-space.

Usage

```
## S4 method for signature 'hmm':
forward(hmm, obs)
## S4 method for signature 'hmm':
backward(hmm, obs)
```

Arguments

hmm	An object of class hmm or one of its subclasses representing the hidden Markov model.
obs	A vector containing the observation sequence.

Value

backward returns the $N \times T$ matrix of (log transformed) backward variables, where N is the number of states of hmm and T is the length of obs.

forward returns a list with components

logProb	log[P(obs|hmm)]
alpha.scaled	The matrix of log transformed forward variables. This has the same dimensions as the matrix returned by backward

Author(s)

Peter Humburg

References

Rabiner, L. R. 1989 A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, **77**(2), 257–286.

See Also

hmm

Examples

```
## create two state HMM with t distributions
state.names <- c("one","two")
transition <- c(0.1, 0.02)
location <- c(1, 2)
scale <- c(1, 1)
df <- c(4, 6)
model <- getHMM(list(a=transition, mu=location, sigma=scale, nu=df),
    state.names)

## obtain observation sequence from model
obs <- sampleSeq(model, 100)

## calculate the probability of the observation given the model
fwd <- forward(model, obs)
fwd$logProb

## compute posterior probabilities
bwd <- backward(model,obs)
post <- bwd + fwd$alpha.scaled
post <- post - apply(post,2,logSum)

## get sequence of most likely states
state.seq <- state.names[apply(post,2,which.max)]
```

---

generate.data *Generate Simulated Dataset*

Description

Generate simulated data based on real data and the results of a previous analysis.

Usage

```
generate.data(data, group, pos.range = c(1, 10),
    num.seq = 100, gap = 35, split.gap = 1000, min.len = 2)
```

Arguments

data	A data.frame with information about genomic coordinates of probes (chromosome and position) in the first two columns. Subsequent columns contain probe measurements of individual samples.
group	Information that can be used to assign probes to one of two classes. Either a logical vector or the name of a GFF file. In the later case all probes in annotated regions are considered to be 'positive'.
pos.range	Indicates how many positive regions should be generated for each observation sequence. The actual number for each sequence is sampled uniformly from the indicated range of values.
num.seq	Number of observation sequences to generate.
gap	Gap between probes. Used to generate artificial probe coordinates.
split.gap	Gap between sequences.
min.len	Minimum number of probes per region.

Value

A list with components

observation	A data.frame with the same format as data.
regions	A list of state sequences.

Author(s)

Peter Humburg

Examples

---

getHMM *Create HMM from Parameter Values*

Description

Create a two state HMM with t emission distributions from a list of parameters.

Usage

```
getHMM(params, snames)
```

Arguments

params	A list with components mu, sigma and nu, each a vector with two elements. They provide values for the location parameter, scale parameter and degrees of freedom for both states. Component a is a vector of length two providing the off diagonal elements of the transition probability matrix.
snames	Two character strings that should be used as state names.

Value

Object of class contHMM with the parameters provided.

Author(s)

Peter Humburg

See Also

hmm.setup

Examples

```
## create two state HMM with t distributions
state.names <- c("one","two")
transition <- c(0.035, 0.01)
location <- c(1, 2)
scale <- c(1, 1)
df <- c(4, 6)
hmm <- getHMM(list(a=transition, mu=location, sigma=scale, nu=df),
    state.names)
```

---

gff2index *Extract Probe Calls from GFF File*

Description

Creates a logical vector indicating probes in annotated regions from information in a GFF formatted file.

Usage

```
gff2index(gff, pos)
```

Arguments

gff	GFF file (see Details).
pos	A data.frame with chromosome names in its first column and probe positions in the second column.

Details

The GFF file can be provided in several ways. Either as data.frame, as a character string providing the name of a GFF file or as a connection object pointing to a GFF file.

Value

A logical vector with one entry for each probe in pos. TRUE indicates probes that are inside a region that is annotated in the provided GFF file.

Author(s)

Peter Humburg

References

GFF specification: http://www.sanger.ac.uk/Software/formats/GFF/GFF_Spec.shtml

See Also

region.position and reg2gff for the reverse operation.

Examples

```
## create two state HMM with t distributions
state.names <- c("one","two")
transition <- c(0.1, 0.02)
location <- c(1, 2)
scale <- c(1, 1)
df <- c(4, 6)
model <- getHMM(list(a=transition, mu=location, sigma=scale, nu=df),
    state.names)

## obtain observation sequence from model
obs <- sampleSeq(model, 100)

## make up some genomic probe coordinates
pos <- data.frame(chromosome = rep("chr1", times = 100),
    position = seq(1, 4000, length = 100))

## compute most likely state sequence for obs
vit.res <- viterbi(model, obs)

## find regions attributed to state "two"
reg.pos <- region.position(vit.res$stateSeq, region="two")

## calculate posterior probability for state "two"
post <- posterior(obs, model, log=FALSE)[1,]

## create gff annotations
gff <- reg2gff(reg.pos, post, pos)

## extract probe calls from gff annotation
probe.idx <- gff2index(gff, pos)
```

---

hmm-class *Class "hmm"*

Description

Virtual base class for HMMs.

Objects from the Class

Do not create objects of this class directly. Instead use objects of derived classes like "contHMM".

Slots

transition.matrix: Object of class "matrix", storing the transition probabilities of the Markov chain.

emission: Object of class "list" containing objects of class "dist" to represent emission distributions for each state.

init: Object of class "numeric". The initial state distribution of the Markov chain.

Methods

baumWelch signature(hmm = "hmm", obs = "list"): Baum-Welch algorithm to estimate model parameters.

viterbiTraining signature(hmm = "hmm", obs = "list"): Viterbi training to estimate model parameters.

viterbi signature(hmm = "hmm"): Viterbi algorithm to calculate most likely state sequence.

backward signature(hmm = "hmm"): Computing of backward variables.

forward signature(hmm = "hmm"): Computing of forward variables.

states signature(hmm = "hmm"): Returns state names.

Author(s)

Peter Humburg

See Also

contHMM, baumWelch, viterbiTraining, viterbi, forward, backward, states

Examples

```
showClass("hmm")
```

---

hmm.setup *Create HMM from Initial Parameter Estimates Obtained from Data*

Description

Convenient way to obtain initial parameter estimates from data.

Usage

```
hmm.setup(data, state = c("enriched", "non-enriched"),
    probe.region = 35, frag.size = 1000, pos.state = 1,
    em.type = "tDist", max.prob = 1, df = 9)
```

Arguments

data	Observation sequence. This can be either a single sequence or a list of sequences.
state	Vector of state names for HMM.
probe.region	Length of genomic region represented by one probe (on average).
frag.size	Expected size of ChIP fragments.
pos.state	Index of state which is considered to represent 'positive' result.
em.type	Character string identifying type of emission distribution to be used. Currently only "tDist" is supported.
max.prob	Maximum probability allowed in transition matrix. Setting this to less than 1 ensures that there are no null transitions.
df	Degrees of freedom for emission distributions.

Details

The parameter estimates are obtained by first clustering the observations, then the mean and variance of the resulting groups are used together with cluster size, expected fragment size and probe density to generate initial values for model parameters.

The parameter values generated by this procedure are only a rough guess and have to be optimised before the model is used for data analysis.

Value

Object of class contHMM.

Note

This method currently only supports two state HMMs with t distributions.

Author(s)

Peter Humburg

See Also

contHMM, getHMM, tDist, viterbiEM

Examples

```
## create two state HMM with t distributions
state.names <- c("one","two")
transition <- c(0.035, 0.01)
location <- c(-1, 2)
scale <- c(1, 1)
df <- c(4, 6)
hmm <- getHMM(list(a=transition, mu=location, sigma=scale, nu=df),
    state.names)

## obtain observation sequence from model
obs <- sampleSeq(hmm, 500)

## build model from data
model <- hmm.setup(obs, state = c("one", "two"),df=5)
```

internals *Internal Functions*

Description

These functions should not be called directly.

Author(s)

Peter Humburg

logSum *Calculate log(x + y) from log(x) and log(y)*

Description

Given $\log(x)$ and $\log(y)$ this function calculates $\log(x + y)$ using the identity

$$\log(x + y) = \log(x) + \log\left(1 + e^{\log(y) - \log(x)}\right)$$

Usage

```
logSum(x, y = NULL, base = 0)
```

Arguments

x	Numeric vector or matrix.
y	Numeric vector or matrix of same dimensions as x or missing.
base	Base of the logarithm to be used. If base is 0 the natural logarithm is used.

Details

If y is missing the function is applied recuresively to all elements of x, i.e., all elements of x are added up. If both x and y are given they are added element wise.

Value

If only x is given a scalar is returned, representing the sum of all elements of x. Otherwise a vector or matrix of the same dimensions as x and y.

Note

This function is useful for cases where x and y cannot be represented accurately by machine numbers but log(x) and log(y) can.

Author(s)

Peter Humburg

Examples

```
x <- 1:4
y <- 4:1
## calculate sum of x an y directly
xy.s <- x + y
## and after log transformation
xy.ls <- logSum(log(x),log(y))
## errors are small:
err <- xy.ls - log(xy.s)
```

---

plot *Plotting of "contDist" Objects*

Description

Functions for plotting of contDist objects, either directly or as part of contHMM objects.

Usage

```
## S3 method for class 'contDist':
plot(x, step.size = 0.01, new.plot = TRUE, weight = 1, ...)
## S3 method for class 'tDist':
plot(x, step.size = 0.01, new.plot = TRUE, weight = 1, ...)
## S3 method for class 'contHMM':
plot(x, ...)
```

Arguments

`x`	Object of class contDist or contHMM.
`step.size`	Numeric value indicating the distance between points at which the density function is evaluated.
`new.plot`	If this is TRUE (the default) a new plot is created, otherwise graph of the density function is added to the current plot.
`weight`	Weighting factor. The density function will be scaled by this factor. This is useful when plotting mixture components.
`...`	Additional arguments to be passed to plot

Value

These functions are called for their side effect.

Author(s)

Peter Humburg

See Also

contDist, contHMM

Examples

```
## create two state HMM with t distributions
state.names <- c("one","two")
transition <- c(0.1, 0.02)
location <- c(1, 2)
scale <- c(1, 1)
df <- c(4, 6)
model <- getHMM(list(a=transition, mu=location, sigma=scale, nu=df),
    state.names)

## plot emission distributions of HMM
plot(model)

## now plot density functions separately
par(mfrow=c(1,1))
plot(model@emission$one)
plot(model@emission$two, new.plot=FALSE, lty=2)
legend("topleft", legend=states(model), lty=1:2)
```

---

posterior *Calculate Posterior Probability for States of HMM*

Description

For each state of an HMM the posterior probability that this state produced a given observation is calculated.

Usage

```
posterior(data, hmm, log = TRUE)
```

Arguments

data	Vector with observation sequence.
hmm	Object of class hmm.
log	Logical indicating whether the logarithm of the posterior probability should be returned.

Details

Regardless of the value of log the computation is carried out in log space. If log = FALSE the result is transformed back to linear space before it is returned.

Value

A matrix with as many rows as hmm has states and one column for each entry in data.

Author(s)

Peter Humburg

See Also

forward, backward

Examples

```
## create two state HMM with t distributions
state.names <- c("one","two")
transition <- c(0.1, 0.02)
location <- c(1, 2)
scale <- c(1, 1)
df <- c(4, 6)
model <- getHMM(list(a=transition, mu=location, sigma=scale, nu=df),
    state.names)

## obtain observation sequence from model
obs <- sampleSeq(model, 100)

## calculate posterior probability for state "one"
post <- posterior(obs, model, log=FALSE)

## get sequence of individually most likely states
state.seq <- apply(post,2,max)
state.seq <- states(model)[state.seq]
```

---

reg2gff *Converting Information about Enriched Regions into GFF Format*

Description

Creates a GFF formatted file with information about enriched regions.

Usage

```
reg2gff(regions, post, probe.pos, file = NULL, score.fun = mean,
    source = "tHMM", feature.type = "posterior_prob",
    class = "ChIP_region", name = "tHMM")
```

Arguments

regions	A matrix indicating enriched regions (see region.position).
post	A vector with posterior probabilities for each probe.
probe.pos	A data.frame with columns "chromosome" and "position" providing genomic coordinates for each probe.
file	Name of GFF file to create.
score.fun	Function used to calculate score of enriched regions.
source	Entry for 'source' field of GFF file.
feature.type	Entry for 'feature' field of GFF file.
class	Class of documented feature. This is used in the 'attribute' field together with name.
name	Name of documented feature. This is used in the 'attribute' field together with class.

Details

`post` should provide scores for each probe. These scores are then summarised for each region by applying `score.fun` to the probe scores in each region. The default for `score.fun` is `mean` but any function that accapts a single numeric vector as its argument and returns a scalar can be used.

Value

A GFF formated `data.frame` is returned invisibly. Usually this function is called for its side effect.

Author(s)

Peter Humburg

References

GFF specification: http://www.sanger.ac.uk/Software/formats/GFF/GFF_Spec.shtml

See Also

`region.position`

Examples

```
## create two state HMM with t distributions
state.names <- c("one","two")
transition <- c(0.1, 0.2)
location <- c(1, 2)
scale <- c(1, 1)
df <- c(4, 6)
model <- getHMM(list(a=transition, mu=location, sigma=scale, nu=df),
    state.names)

## obtain observation sequence from model
obs <- sampleSeq(model, 100)

## make up some genomic probe coordinates
pos <- data.frame(chromosome = rep("chr1", times = 100),
    position = seq(1, 4000, length = 100))

## compute most likely state sequence for obs
vit.res <- viterbi(model, obs)

## find regions attributed to state "one"
reg.pos <- region.position(vit.res$stateSeq, region="one")

## calculate posterior probability for state "one"
post <- posterior(obs, model, log=FALSE)[1,]

## create gff annotations
gff <- reg2gff(reg.pos, post, pos)
```

region.length *Determine Length of Positive and Negative Regions*

Description

Given a logical vector indicating positive and negative probes this function returns a list with components 'positive' and 'negative' providing length information for positive and negative regions.

Usage

```
region.length(probes, min.len=1)
```

Arguments

probes	A logical vector indicating the position of enriched and non-enriched probes with TRUE and FALSE respectively.
min.len	The minimal number of consecutive probes required to form a region.

Value

A list with components positive and negative, each containing a numeric vector with the length of identified regions.

Author(s)

Peter Humburg

See Also

region.position

Examples

```
## create random probe calls
probes <- sample(c(TRUE,FALSE), 200, replace=TRUE)

## find length of all regions that contain at least two probes
reg.len <- region.length(probes, min.len=2)
```

region.position *Identify Enriched Regions*

Description

After calling individual probes enriched or non-enriched this function can be used to combine probes into enriched regions.

Usage

```
region.position(probe.calls, region = TRUE)
```

Arguments

probe.calls	A vector of probe calls.
region	Probe call that should be combined into regions.

Value

A matrix with two rows and as many columns as there are enriched regions. The first row gives the index of the first probe of each region the second row the index of the last probe inside each region.

Author(s)

Peter Humburg

Examples

```
## create two state HMM with t distributions
state.names <- c("one","two")
transition <- c(0.1, 0.02)
location <- c(1, 2)
scale <- c(1, 1)
df <- c(4, 6)
model <- getHMM(list(a=transition, mu=location, sigma=scale, nu=df),
    state.names)

## obtain observation sequence from model
obs <- sampleSeq(model, 100)

## compute most likely state sequence for obs
vit.res <- viterbi(model, obs)

## find regions attributed to state "one"
reg.pos <- region.position(vit.res$stateSeq, region="one")
```

---

remove.short *Post-Processing of "tileHMM" Results*

Description

Remove short regions that are likely to be spurious.

Usage

```
remove.short(regions, post, probe.pos, min.length = 1000,
    min.score = 0.8, summary.fun = mean)
```

Arguments

regions	A matrix with information about the location of enriched regions.
post	A numeric vector with the posterior probability of ChIP enrichment for each probe.
probe.pos	A data frame with columns 'chromosome' and 'position' providing genomic coordinates for each probe.

min.length	Minimum length of enriched regions (see Details).
min.score	Minimum score for enriched regions (see Details).
summary.fun	Function used to summarise posterior probe probabilities into region scores.

Details

All regions that are shorter than min.length **and** have a score of less than min.score will be removed. To filter regions based on only one of these values set the other one to 0.

Region scores are calculated based on posterior probe probabilities. The summary function used should accept a single numeric argument and return a numeric vector of length 1. If the probabilities in post are log transformed they will be transformed back to linear space before they are summarised for each region.

Value

A matrix with two rows and one column for each remaining region.

Author(s)

Peter Humburg

See Also

region.position

Examples

```
## create two state HMM with t distributions
state.names <- c("one","two")
transition <- c(0.1, 0.1)
location <- c(1, 2)
scale <- c(1, 1)
df <- c(4, 6)
model <- getHMM(list(a=transition, mu=location, sigma=scale, nu=df),
    state.names)

## obtain observation sequence from model
obs <- sampleSeq(model, 500)

## make up some genomic probe coordinates
pos <- data.frame(chromosome = rep("chr1", times = 500),
    position = seq(1, 18000, length = 500))

## calculate posterior probability for state "one"
post <- posterior(obs, model, log=FALSE)

## get sequence of individually most likely states
state.seq <- apply(post, 2, which.max)
state.seq <- states(model)[state.seq]

## find regions attributed to state "one"
reg.pos <- region.position(state.seq, region="one")

## remove short and unlikely regions
reg.pos2 <- remove.short(reg.pos, post, pos, min.length = 200,
```

```
    min.score = 0.8)
```

`sampleObs` *Sample Observations from Probability Distribution*

Description

Draws a sample of given size from a discrete or continuous probability distribution.

Usage

```
## S4 method for signature 'discDist, numeric':
sampleObs(dist, size, ...)
## S4 method for signature 'tDist, numeric':
sampleObs(dist, size, ...)
```

Arguments

`dist`	Object of class `discDist` or `tDist`. The probability distribution to use.
`size`	Sample size.
`...`	Additional arguments.

Value

Vector of sampled values.

Author(s)

Peter Humburg

See Also

`dist`

Examples

```
## sampling from a t distribution
tdist <- new("tDist",mean=0,var=1,df=3)
obs <- sampleObs(tdist,100)

## sampling from a discrete distribution
nucleo <- new("discDist",alpha=c("A","C","G","T"),prob=rep(0.25,times=4))
dna <- sampleObs(nucleo,100)
```

sampleSeq *Generate Observation Sequence from HMM*

Description

Generates an observation sequence according to a model given as object of class hmm. Optionally the undelying state sequence is returned together with the observations.

Usage

```
## S4 method for signature 'contHMM, numeric':
sampleSeq(hmm, size, return.states=FALSE)
```

Arguments

hmm	Object of class contHMM.
size	Numeric value indicating the desired length of the observation sequence.
return.states	Logical indicating whether the underlying state sequence should be returned together with the observation sequence.

Value

If return.states is FALSE (the default) a vector of length size with observations sampled from hmm. If return.states is TRUE a list with components

states	Character vector with state sequence that was used to generate the observations.
observation	Vector with observations sampled from hmm.

Author(s)

Peter Humburg

See Also

hmm, states

Examples

```
## create two state HMM with t distributions
state.names <- c("one","two")
transition <- c(0.035, 0.01)
location <- c(1, 2)
scale <- c(1, 1)
df <- c(4, 6)
hmm1 <- getHMM(list(a=transition, mu=location, sigma=scale, nu=df),
    state.names)

## generate observation sequence from model
obs <- sampleSeq(hmm1, 100)
```

shrinkt.st *Calculate 'Shrinkage t' Statistic*

Description

Calculate the 'shrinkage *t*' statistic (Opgen-Rhein and Strimmer, 2007).

Usage

```
shrinkt.st(X, L, h0.mean = 0, ...)
```

Arguments

X	Data matrix. Rows correspond to variables (probes) and columns to samples.
L	Design vector. Indicating treatment (1) and control (2) samples. If no control samples are given a one sample *t* test is carried out.
h0.mean	If no control samples are provided the treatment mean is compared to this value in a one sample *t* test.
...	Further arguments to be passed to and from other methods.

Value

An object of class "shrinkage" containing the test statistics.

Note

This method uses the shrinkt.stat method from package st.

Author(s)

Peter Humburg

References

Opgen-Rhein, R., and K. Strimmer 2007. Accurate ranking of differentially expressed genes by a distribution-free shrinkage approach. *Statist. Appl. Genet. Mol. Biol.* **6**:9. http://www.bepress.com/sagmb/vol6/iss1/art9/

See Also

shrinkt.stat

Examples

```
## generate some data
X <- matrix(nrow = 100, ncol = 6)
X[ , 1:3] <- apply(X[ , 1:3], 1, rnorm,
    mean = rnorm(1, mean = 2, sd = 2), sd = rchisq(1, df = 2))
X[ , 4:6] <- apply(X[ , 4:6], 1, rnorm,
    mean = rnorm(1, mean = 0, sd = 1), sd = rchisq(1, df = 2))
L <- c(1, 1, 1, 2, 2, 2)
```

```
## compute shrinkage t statistic
st.stat <- shrinkt.st(X, L)
```

`simChIP` *Simulated ChIP-on-Chip Data*

Description

A simulated ChIP-on-chip dataset. This is part of the data used in a simulation study by Humburg *et al.* (2007), it is based on data published by Zhang *et al.* (2007).

Usage

```
data(simChIP)
```

Format

A data frame with 210828 probe measurements under three different conditions with four replicates each.

`chromosome` The chromosome targeted by probe. Here always 'chr1'.

`position` Position of central base.

`input_1` Genomic DNA sample 1.

`input_2` Genomic DNA sample 2.

`input_3` Genomic DNA sample 3.

`input_4` Genomic DNA sample 4.

`h3_1` Histone H3 ChIP sample 1.

`h3_2` Histone H3 ChIP sample 2.

`h3_3` Histone H3 ChIP sample 3.

`h3_4` Histone H3 ChIP sample 4.

`h3k27_1` Histone H3K27me3 ChIP sample 1.

`h3k27_2` Histone H3K27me3 ChIP sample 2.

`h3k27_3` Histone H3K27me3 ChIP sample 3.

`h3k27_4` Histone H3K27me3 ChIP sample 4.

Source

Humburg, P. and Bulger, D. and Stone, G. 2008 Parameter estimation for robust HMM analysis of ChIP-chip data. *unpublished*

References

Zhang, X. and Clarenz, O. and Cokus, S. and Bernatavichute, Y. V. and Goodrich, J. and Jacobsen S. E. 2007 Whole-genome analysis of histone H3 lysine 27 trimethylation in Arabidopsis. *PLoS Biology*, **5**(5).

Examples

```
data(simChIP)
## Not run:
## scatter plots
library(geneplotter)
simChIP[ , 3:14] <- logb(simChIP[ , 3:14], 2)
par(mfrow=c(2,2))
smoothScatter(simChIP[ , 7], simChIP[ , 11], xlab = "H3 sample 1",
    ylab = "H3K27me3 sample 1")
smoothScatter(simChIP[ , 8], simChIP[ , 12], xlab = "H3 sample 2",
    ylab = "H3K27me3 sample 2")
smoothScatter(simChIP[ , 9], simChIP[ , 13], xlab = "H3 sample 3",
    ylab = "H3K27me3 sample 3")
smoothScatter(simChIP[ , 10], simChIP[ , 14], xlab = "H3 sample 4",
    ylab = "H3K27me3 sample 4")
## End(Not run)
```

---

states *State Names of Hidden Markov Model*

Description

Returns a vector of state names for given HMM.

Usage

```
## S4 method for signature 'hmm':
states(hmm, ...)
```

Arguments

hmm	Object of class hmm
...	Additional arguments.

Value

A character vector with as many entries as hmm has states. Each entry represents the name of the corresponding state.

Author(s)

Peter Humburg

See Also

hmm

Examples

```
## create two state HMM with t distributions
state.names <- c("one","two")
transition <- c(0.1, 0.02)
location <- c(1, 2)
scale <- c(1, 1)
df <- c(4, 6)
model <- getHMM(list(a=transition, mu=location, sigma=scale, nu=df),
    state.names)

## obtain observation sequence from model
obs <- sampleSeq(model, 100, return.states=TRUE)

## compute most likely state sequence for obs
## return sequence of state indices instead of names
vit.res <- viterbi(model, obs$observation, names=FALSE)

## get sequence of state names via call to 'states'
state.seq <- states(model)[vit.res$stateSeq]
```

---

tDist-class *Class "tDist"*

Description

This class provides objects representing scaled *t* distributions.

Objects from the Class

Objects can be created by calls of the form `new("tDist", mean, var, df)`.

`mean`: Location parameter.

`var`: Scale parameter.

`df`: Degrees of freedom.

The distribution has density

$$f(x; \texttt{mean}, \texttt{var}, \texttt{df}) = \frac{\Gamma\left(\frac{\texttt{df}+1}{2}\right)}{\sqrt{\texttt{var} \times \pi \times \texttt{df}}\,\Gamma\left(\frac{\texttt{df}}{2}\right)} \times \frac{1}{\sqrt{\left(1 + \frac{(x-\texttt{mean})^2}{\texttt{var}\times\texttt{df}}\right)^{\texttt{df}+1}}}$$

Slots

`components`: Object of class `"matrix"` with columns `weight`, `mean`, `variance`, and `df`. For objects of class `"tDist"` this matrix has only one row and `weight` is always 1.

Extends

Class `"contDist"`, directly. Class `"dist"`, by class "contDist", distance 2.

Methods

sampleObs `signature(dist = "tDist", size = "numeric")`: Sample `size` observations from `dist`.

Author(s)

Peter Humburg

See Also

`contHMM`

Examples

```
showClass("tDist")
```

`tileHMM-package` *Hidden Markov Models for ChIP-on-Chip Analysis*

Description

This package provides methods and classes to build HMMs that are suitable for the analysis of ChIP-on-chip data. The provided parameter estimation methods include the Baum-Welch algorithm and Viterbi training as well as a combination of both. The latter provides results identical to the Baum-Welch algorithm but is conciderably faster.

Details

Package:	tileHMM
Type:	Package
Version:	1.0
Date:	2007-11-13
License:	GPL

Hidden Markov models are represented as objects of class `hmm` or derived classes. Function `getHMM` provides an easy to use interface to create `contHMM` objects with emission distributions of class `tDist` from a set of parameters. Function `hmm.setup` can be used to create HMMs with initial parameter estimates obtained from data.

To optimise initial parameter estimates Viterbi training and the Baum-Welch algorithm are provided by this package. Function `viterbiEM` provides a convenient way to use a combination of both methods.

Author(s)

Peter Humburg

Maintainer: Peter Humburg ⟨Peter.Humburg@csiro.au⟩

References

Humburg, P. and Bulger, D. and Stone, G. 2008 Parameter estimation for robust HMM analysis of ChIP-chip data. *unpublished*

See Also

Classes provided by this package: hmm, contHMM, dist, discDist, contDist, tDist

Important functions: viterbiEM, baumWelch, viterbiTraining

`viterbiEM` *Efficient Estimation of HMM Parameters*

Description

Uses a combination of Viterbi training and Baum-Welch algorithm to estimate parameters for a hidden Markov model.

Usage

```
viterbiEM(hmm, data, max.iter = c(5, 15), eps = 0.01,
    verbose = 0, ...)
```

Arguments

hmm	Object of class hmm. This is used as starting point for the optimisation procedure.
data	A list of observation sequences.
max.iter	Maximum number of iterations (see Details).
eps	Minimum change in log-likelihood between iterations (see Details).
...	Additional arguments to be passed to viterbiTraining and baumWelch (see Details).
verbose	Level of verbosity. Higher numbers produce more status messages.

Details

The values of arguments max.iter and eps can have either one or two elements. In the latter case the first element is used for viterbiTraining and the second one for baumWelch.

Additional arguments can be passed to viterbiTraining and baumWelch by using arguments of the form viterbi = list(a = a.value) and baumWelch = list(b = b.value) respectively. All other arguments are passed on to both functions.

Value

An object of class hmm with optimised parameter estimates.

Author(s)

Peter Humburg

References

Humburg, P. and Bulger, D. and Stone, G. 2008 Parameter estimation for robust HMM analysis of ChIP-chip data. *unpublished*

See Also

baumWelch, viterbiTraining, hmm.setup

Examples

```
## create two state HMM with t distributions
state.names <- c("one","two")
transition <- c(0.035, 0.01)
location <- c(1, 2)
scale <- c(1, 1)
df <- c(4, 6)
hmm1 <- getHMM(list(a=transition, mu=location, sigma=scale, nu=df),
    state.names)

## generate observation sequences from model
obs.lst <- list()
for(i in 1:50) obs.lst[[i]] <- sampleSeq(hmm1, 100)

## fit an HMM to the data (with fixed degrees of freedom)
hmm2 <- hmm.setup(obs.lst, state=c("one","two"), df=5)
hmm2.fit <- viterbiEM(hmm2, obs.lst, max.iter=c(5,15), verbose=2, df=5)
```

---

viterbi *Calculate Most Likely State Sequence Using the Viterbi Algorithm*

Description

The Viterbi algorithm computes the most likely sequence of states given an HMM and an observation sequence.

Usage

```
## S4 method for signature 'hmm':
viterbi(hmm, obs, names=TRUE)
```

Arguments

hmm	Object of class hmm.
obs	A vector containing the observation sequence.
names	Logical indicating whether state names should be returned. If TRUE (the default) the returned sequence consists of state names, otherwise the state index is returned instead.

Value

A list with components

stateSeq	Most likely state sequence.
logProb	The probability of stateSeq given hmm and obs.
matrix	The dynamic programming matrix.

Author(s)

Peter Humburg

References

Viterbi, A. J. 1967 Error bounds for convolutional codes and an assymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory*, **13**, 2600–269.

See Also

viterbiTraining

Examples

```
## create two state HMM with t distributions
state.names <- c("one","two")
transition <- c(0.1, 0.02)
location <- c(1, 2)
scale <- c(1, 1)
df <- c(4, 6)
model <- getHMM(list(a=transition, mu=location, sigma=scale, nu=df),
    state.names)

## obtain observation sequence from model
obs <- sampleSeq(model, 100, return.states=TRUE)

## compute most likely state sequence for obs
vit.res <- viterbi(model, obs$observation)

## how well did we do?
sum(vit.res$stateSeq == obs$states)/length(vit.res$stateSeq)
```

---

viterbiTraining *Estimate HMM Parameters Using Viterbi Training*

Description

Viterbi training is a faster but less reliable alternative to Baum-Welch for parameter estimation.

Usage

```
## S4 method for signature 'hmm, list':
viterbiTraining(hmm, obs, max.iter=10, eps=0.01,
        df=NULL, trans.prior=NULL, init.prior=NULL, keep.models=NULL, verbose=1)
```

Arguments

hmm	Object of class hmm.
obs	List of observation sequences.
max.iter	Maximum number of iterations.
eps	Minimum change in log likelihood between successive iterations.
df	If this is NULL the degrees of freedom for the t distributions are e stimated from the data. Otherwise they are set to df.
trans.prior	Prior distribution of transition probabilities. A prior can be specified either by providing a matrix with transition probabilities or by setting trans.prior=TRUE. In the latter case the initial parameter estimates are used as prior. If trans.prior is NULL (the default) no prior is used.
init.prior	Prior distribution of initial state probabilities. A prior can be specified either by providing a vector with initial state probabilities or by setting init.prior=TRUE. In the latter case the initial parameter estimates are used as prior. If init.prior is NULL (the default) no prior is used.
keep.models	A character string interpreted as a file name. If keep.models is not NULL the models produced during the parameter estimation procedure are saved to a file.
verbose	Level of verbosity. Allows some control over the amount of output printed to the console.

Value

Object of class hmm with the pest parameter estimates (in terms of likelihood) found during the fitting procedure.

Author(s)

Peter Humburg

References

Juang, B.-H. and Rabiner, L. R. 1990 A segmental k-means algorithm for estimating parameters of hidden Markov models. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, **38**(9), 1639–1641.

See Also

viterbi, baumWelch, viterbiEM, hmm.setup

Examples

```
## create two state HMM with t distributions
state.names <- c("one","two")
transition <- c(0.035, 0.01)
location <- c(1, 2)
scale <- c(1, 1)
df <- c(4, 6)
hmm1 <- getHMM(list(a=transition, mu=location, sigma=scale, nu=df),
    state.names)

## generate observation sequences from model
```

```
obs.lst <- list()
for(i in 1:50) obs.lst[[i]] <- sampleSeq(hmm1, 100)

## fit an HMM to the data (with fixed degrees of freedom)
hmm2 <- hmm.setup(obs.lst, state=c("one","two"), df=5)
hmm2.fit <- viterbiTraining(hmm2, obs.lst, max.iter=20, df=5, verbose=1)

## fit an HMM to the data, this time estimating the degrees of freedom
hmm3.fit <- viterbiTraining(hmm2, obs.lst, max.iter=20, verbose=1)
```

Index