Package 'condmixt'

May 11, 2020

Type Package

Title Conditional Density Estimation with Neural Network Conditional Mixtures

Version 1.1

Date 2020-05-09

Author Julie Carreau

Maintainer Julie Carreau <julie.carreau@ird.fr>

Description Neural network conditional mixtures are mixture models whose parameters are predicted by a neural network. The mixture model can thus change its parameters in response to changes in predictive covariates. Mixtures included are gaussian, log-normal and hybrid Pareto mixtures. The latter relies on the generalized Pareto distribution to account for the presence of large extreme events. The unconditional mixtures are also available.

Depends evd

License GPL-2

LazyLoad yes

NeedsCompilation yes

Repository CRAN

Date/Publication 2020-05-11 09:10:08 UTC

R topics documented:

`condmixt-package` *Conditional Density Estimation with Neural Network Conditional Mixtures*

Description

Neural network conditional mixtures are mixtures whose parameters depends on explanatory variables through a neural network. In other words, for a given set of values of explanatory variables, a neural network will compute the corresponding mixture parameters. Thus, the parameters of the mixture and hence the shape of its density is modified by the values of the explanatory variables. There are two hyper-parameters, the number of hidden units of the neural network and the number of components of the mixture, which control the complexity of the conditional mixtures. In this package, a number of types of conditional mixtures are implemented, which differ in their type of components and which allow the possibility to include a discrete component in the mixture.

Details

Package:	condmixt
Type:	Package
Version:	1.0
Date:	2012-04-06
License:	GPL-2
LazyLoad:	yes

Functions for conditional mixtures with - hybrid Pareto components start with `condhparetomixt` - Gaussian components start with `condgaussmixt` - Log-Normal components start with `condlognormixt`

If a discrete dirac component at zero is added, for instance to model the presence of zeros for "no rain" events in a rainfall time-series, then functions related to such mixtures start with something like `condhparetomixt.dirac` depending on the type of components for the continuous part.

One special mixture, which has only two-components (a discrete, dirac or Bernoulli, component and a Gamma component) is implemented under the name `condbergamixt`. It has been introduced in Williams(1998).

Finally, hybrid Pareto components have a tail index parameter which might be difficult to estimate, specially in the conditional, non-stationnary case. To alleviate this issue, a penalty term might be added to the log-likelihood in order to guide maximum likelihood estimation of the tail indexes. The functions which employ a tail penalty term have a name ending with `tailpen`.

The goal of the tail penalty for hybrid Pareto mixtures is to include a priori information which in our case is that only a few mixture components have a heavy tail index which should approximate the tail of the underlying distribution while most other mixture components have a light tail and aim at modelling the central part of the underlying distribution.

The penalty term is given by the logarithm of the following two-component mixture, as a function of a tail index parameter xi :

w beta exp(-beta xi) + (1-w) exp(-(xi-mu)^2/(2 sigma^2))/(sqrt(2 pi) sigma)

where the first term is the prior on the light tail component and the second term is the prior on the heavy tail component.

Author(s)

Julie Carreau

Maintainer: Julie Carreau <julie.carreau@univ-montp2.fr>

References

Bishop, C. (1995), Neural Networks for Pattern Recognition, Oxford

Carreau J. and Vrac, M. (2011) Stochastic Downscaling of Precipitation with Neural Network Conditional Mixture Models, 47, Water Resources Research

Carreau, J. and Bengio, Y. (2009), A Hybrid Pareto Model for Asymmetric Fat-tailed Data: the Univariate Case, 12, Extremes

Carreau, J. and Bengio, Y. (2009), A Hybrid Pareto Mixture for Conditional Asymmetric Fat-Tailed Distributions, 20, IEEE Transactions on Neural Networks

Williams, M.P. (1998) Modelling Seasonality and Trends in Daily Rainfall Data, 10, Advances in Neural Information and Processing Systems

See Also

`condmixt.init`, `condmixt.nll`, `condmixt.fit`

condgaussmixt *The conditional Gaussian mixture distribution*

Description

Distribution function and density function for the conditional Gaussian mixture without discrete component.

Usage

```
pcondgaussmixt(params, m, y, trunc = TRUE)
dcondgaussmixt(params,m,y,log=FALSE,trunc=TRUE)
```

Arguments

params	m x 3 x n matrix of mixture parameters where n is the number of examples
m	Number of mixture components.
y	Vector of n dependent variables.
log	logical, if TRUE, probabilities p are given as log(p).
trunc	logical, if TRUE (default), the hybrid Pareto density is truncated below zero.

Details

params can be computed by applying condgaussmixt.fwd on the explanatory variables x of dimension d x n associated with y.

Value

Distribution function evaluated at n points for pcondgaussmixt and density function for dcondgaussmixt.

Author(s)

Julie Carreau

References

Bishop, C. (1995), Neural Networks for Pattern Recognition, Oxford

Carreau, J. and Bengio, Y. (2009), A Hybrid Pareto Mixture for Conditional Asymmetric Fat-Tailed Distributions, 20, IEEE Transactions on Neural Networks

See Also

condmixt.nll, condmixt.fwd

Examples

```
# generate train data
ntrain <- 200
xtrain <- runif(ntrain)
ytrain <- rfrechet(ntrain,loc = 3*xtrain+1,scale =
0.5*xtrain+0.001,shape=xtrain+1)
plot(xtrain,ytrain,pch=22) # plot train data
qgen <- qfrechet(0.99,loc = 3*xtrain+1,scale = 0.5*xtrain+0.001,shape=xtrain+1)
points(xtrain,qgen,pch="*",col="orange")

h <- 4 # number of hidden units
m <- 2 # number of components

# initialize a conditional mixture with Gaussian components and a dirac at zero
thetainit <- condgaussmixt.init(1,h,m,ytrain)

# compute mixture parameters
params.mixt <- condgaussmixt.fwd(thetainit,h,m,t(xtrain))

cdf <- pcondgaussmixt(params.mixt,m,ytrain) # compute CDF
```

`condhparetomixt` *The conditional hybrid Pareto mixture distribution*

Description

Distribution function for the conditional hybrid Pareto mixture with and without discrete component at zero and density function for the conditional hybrid Pareto mixture without discrete component.

Usage

```
pcondhparetomixt(params, m, y, trunc = TRUE)
pcondhparetomixt.dirac(params,m,y)
dcondhparetomixt(params,m,y,log=FALSE,trunc=TRUE)
```

Arguments

`params`	m x 4 x n matrix of mixture parameters where n is the number of examples for `condhparetomixt` and (4m+1) x n matrix for `condhparetomixt.dirac`
`m`	Number of mixture components.
`y`	Vector of n dependent variables.
`log`	logical, if TRUE, probabilities p are given as log(p).
`trunc`	logical, if TRUE (default), the hybrid Pareto density is truncated below zero. A mixture with a Dirac component at zero is always truncated below zero.

Details

params can be computed from the forward functions on the explanatory variables x of dimension d x n associated with y : condhparetomixt.fwd and condhparetomixt.dirac.fwd

Value

Distribution function evaluated at n points for pcondhparetomixt and pcondhparetomixt.dirac and density function for dcondhparetomixt.

Author(s)

Julie Carreau

References

Carreau, J. and Bengio, Y. (2009), A Hybrid Pareto Mixture for Conditional Asymmetric Fat-Tailed Distributions, 20, IEEE Transactions on Neural Networks

See Also

condmixt.nll, condmixt.fwd

Examples

```
# generate train data with a mass at zero
ntrain <- 200
xtrain <- runif(ntrain,0,2*pi)
alpha.train <- sin(2*xtrain)/2+1/2
data.train <- rep(0,ntrain)
for (i in 1:ntrain){
  if (sample(c(0,1),1,prob=c(1-alpha.train[i],alpha.train[i]))){
   # rainy day, sample from a Frechet
    data.train[i] <-rfrechet(1,loc=3*sin(2*xtrain[i])+4,scale=1/(1+exp(-(xtrain[i]-1))),
                    shape=(1+exp(-(xtrain[i]/5-2))))
  }
}

plot(xtrain,data.train,pch=20)


h <- 4 # number of hidden units
m <- 2 # number of components

# initialize a conditional mixture with hybrid Pareto components and a dirac at zero
thetainit <- condhparetomixt.dirac.init(1,h,m,data.train)


# compute mixture parameters on test data
params.mixt <- condhparetomixt.dirac.fwd(thetainit,h,m,t(xtrain))
```

```
cdf <- pcondhparetomixt.dirac(params.mixt,m,data.train) # compute CDF on test data
```

`condhparetomixt.cvtrain.tailpen` *Cross-validation of the conditinal mixture with hybrid Pareto components with a tail penalty added to the negative log-likelihood for training.*

Description

K-fold cross-validation is performed in order to select hyper-parameters of the conditional mixture with hybrid Pareto components. A tail penalty is added to the negative log-likelihood in order to guide the tail index parameters estimation. Performance is evaluated by computing the negative log-likelihood, without penalty.

Usage

```
condhparetomixt.cvtrain.tailpen(x, y, hp, nfold = 5, nstart = 1, ...)
```

Arguments

`x`	Matrix of explanatory (independent) variables of dimension d x n, d is the number of variables and n is the number of examples (patterns)
`y`	Vector of n dependent variables
`hp`	Matrix nhp x 7 whose rows represent a set of values for the following hyper-parameters : number of hidden unit, number of component, lambda, w, beta, mu and sigma. The last five hyper-parameters control the tail penalty, see in the Details section below.
`nfold`	Number of folds for the cross-validation, default is 5.
`nstart`	Number of optimization re-starts for each training, default is one. Optimization is re-started from different initial values in order to avoir local minima.
`...`	Extra arguments passed to `nlm`.

Details

The penalty term is given by the logarithm of the following two-component mixture, as a function of a tail index parameter xi :

w beta exp(-beta xi) + (1-w) exp(-(xi-mu)^2/(2 sigma^2))/(sqrt(2 pi) sigma)

where the first term is the prior on the light tail component and the second term is the prior on the heavy tail component.

The extra hyper-parameters for the penalty terms are as follows : - lambda : penalty parameter which controls the trade-off between the penalty and the negative log-likelihood, takes on positive values. If zero, no penalty - w : penalty parameter in [0,1] which is the proportion of components

with light tails, 1-w being the proportion of components with heavy tails - beta : positive penalty parameter which indicates the importance of the light tail components (it is the parameter of an exponential which represents the prior over the light tail components) - mu : penalty parameter in (0,1) which represents the a priori value for the heavy tail index of the underlying distribution - sigma : positive penalty parameter which controls the spread around the a priori value for the heavy tail index of the underlying distribution

Value

Returns a vector of negative log-likelihood values evaluated out-of-sample by cross-validation. Elements in the vector correspond to each set of hyper-parameters evaluated.

Author(s)

Julie Carreau

References

Bishop, C. (1995), Neural Networks for Pattern Recognition, Oxford

Carreau, J. and Bengio, Y. (2009), A Hybrid Pareto Mixture for Conditional Asymmetric Fat-Tailed Distributions, 20, IEEE Transactions on Neural Networks

See Also

condmixt.foldtrain,condmixt.train,condmixt.nll, condmixt.init

Examples

```
n <- 200
x <- runif(n) # x is a random uniform variate
 # y depends on x through the parameters of the Frechet distribution
y <- rfrechet(n,loc = 3*x+1,scale = 0.5*x+0.001,shape=x+1)

plot(x,y,pch=22)
# 0.99 quantile of the generative distribution
qgen <- qfrechet(0.99,loc = 3*x+1,scale = 0.5*x+0.001,shape=x+1)
points(x,qgen,pch="*",col="orange")

# create a matrix with sets of values for the number of hidden units and
# the number of components
hp <- matrix(c(2,4,2,2),nrow=2,ncol=2)

# keep tail penalty parameters constant
hp <- cbind(hp, rep(10,2),rep(0.5,2),rep(20,2),rep(0.2,2),rep(0.5,2))

condhparetomixt.cvtrain.tailpen(t(x), y, hp, nfold = 2, nstart = 2, iterlim=100)
```

condmixt.dirac.negloglike *Negative log-likelihood for conditional mixture with a discrete component at zero.*

Description

The negative log-likelihood is computed from mixture parameters rather than from neural network parameters. The mixture parameters are obtained from the neural network for given explanatory variable values.

Usage

```
condhparetomixt.dirac.negloglike(params, m, y)
condlognormixt.dirac.negloglike(params,m,y)
condgaussmixt.dirac.negloglike(params,m,y)
condbergamixt.negloglike(params,y)
```

Arguments

params	p x n matrix of mixture parameters where n is the number of examples and p = (4m+1) for condhparetomixt.dirac, p = (3m+1) for condgaussmixt.dirac and condlognormixt.dirac and p = 3 for condbergamixt.
m	Number of components in the mixture.
y	Vector of n dependent variables.

Details

params can be computed from the forward functions on the explanatory variables x of dimension d x n associated with y : condhparetomixt.dirac.fwd, condgaussmixt.dirac.fwd (which can be used for conditional mixtures with Log-Normal components) and condbergamixt.fwd

Value

Vector of length n corresponding to the negative log-likelihood evaluated on each example.

Author(s)

Julie Carreau

References

Bishop, C. (1995), Neural Networks for Pattern Recognition, Oxford

Carreau, J. and Bengio, Y. (2009), A Hybrid Pareto Mixture for Conditional Asymmetric Fat-Tailed Distributions, 20, IEEE Transactions on Neural Networks

See Also

condmixt.fwd

Examples

```
# generate train data with a mass at zero
ntrain <- 200
xtrain <- runif(ntrain,0,2*pi)
alpha.train <- sin(2*xtrain)/2+1/2
data.train <- rep(0,ntrain)
for (i in 1:ntrain){
  if (sample(c(0,1),1,prob=c(1-alpha.train[i],alpha.train[i]))){
   # rainy day, sample from a Frechet
    data.train[i] <-rfrechet(1,loc=3*sin(2*xtrain[i])+4,scale=1/(1+exp(-(xtrain[i]-1))),
                    shape=(1+exp(-(xtrain[i]/5-2))))
  }
}

plot(xtrain,data.train,pch=20)


h <- 4 # number of hidden units
m <- 2 # number of components

# initialize a conditional mixture with hybrid Pareto components and a
# dirac at zero
 thetainit <- condhparetomixt.dirac.init(1,h,m,data.train)

# compute mixture parameters
params.mixt <- condhparetomixt.dirac.fwd(thetainit,h,m,t(xtrain))

# compute negative log-likelihood
nll <- condhparetomixt.dirac.negloglike(params.mixt, m, data.train)
```

---

condmixt.fit *Maximum likelihood estimation for conditional mixture parameters*

Description

Performs maximum likelihood estimation of conditional mixture parameters given starting values and the numbers of hidden units and of components, the type of components and the presence of a discrete dirac component on a given data set.

Usage

```
condhparetomixt.fit(theta, h, m, x, y, ...)
condhparetomixt.fit.tailpen(theta,h,m,x,y,lambda=0,w=0.2,beta=50,
                            mu=0.2,sigma=0.2, ...)
```

```
condhparetomixt.dirac.fit.tailpen(theta,h,m,x,y,lambda=0,w=0.2,beta=50,
                                  mu=0.2,sigma=0.2, ...)
condhparetomixt.dirac.fit(theta,h,m,x,y, ...)
condgaussmixt.fit(theta,h,m,x,y, ...)
condgaussmixt.dirac.fit(theta,h,m,x,y, ...)
condlognormixt.fit(theta,h,m,x,y, ...)
condlognormixt.dirac.fit(theta,h,m,x,y, ...)
condbergamixt.fit(theta,h,x,y, ...)
```

Arguments

theta	Vector of neural network parameters
h	Number of hidden units
m	Number of components
x	Matrix of explanatory (independent) variables of dimension d x n, d is the number of variables and n is the number of examples (patterns)
y	Vector of n dependent variables
lambda	penalty parameter which controls the trade-off between the penalty and the negative log-likelihood, takes on positive values. If zero, no penalty
w	penalty parameter in [0,1] which is the proportion of components with light tails, 1-w being the proportion of components with heavy tails
beta	positive penalty parameter which indicates the importance of the light tail components (it is the parameter of an exponential which represents the prior over the light tail components)
mu	penalty parameter in (0,1) which represents the a priori value for the heavy tail index of the underlying distribution
sigma	positive penalty parameter which controls the spread around the a priori value for the heavy tail index of the underlying distribution
...	optional arguments for the optimizer nlm

Details

condhparetomixt indicates a mixture with hybrid Pareto components, condgaussmixt for Gaussian components, condlognormixt for Log-Normal components, condbergam for a Bernoulli-Gamma two component mixture, tailpen indicates that a penalty is added to the log-likelihood to guide the tail index parameter estimation, dirac indicates that a discrete dirac component at zero is included in the mixture

In order to drive the tail index estimation, a penalty is introduced in the log-likelihood. The goal of the penalty is to include a priori information which in our case is that only a few mixture components have a heavy tail index which should approximate the tail of the underlying distribution while most other mixture components have a light tail and aim at modelling the central part of the underlying distribution.

The penalty term is given by the logarithm of the following two-component mixture, as a function of a tail index parameter xi :

w beta exp(-beta xi) + (1-w) exp(-(xi-mu)^2/(2 sigma^2))/(sqrt(2 pi) sigma)

where the first term is the prior on the light tail component and the second term is the prior on the heavy tail component.

Value

Returns a vector of neural network parameters corresponding to the maximum likelihood estimation.

Author(s)

Julie Carreau

References

Bishop, C. (1995), Neural Networks for Pattern Recognition, Oxford

Carreau, J. and Bengio, Y. (2009), A Hybrid Pareto Mixture for Conditional Asymmetric Fat-Tailed Distributions, 20, IEEE Transactions on Neural Networks

See Also

condmixt.nll, condmixt.init

Examples

```
n <- 200
x <- runif(n) # x is a random uniform variate
# y depends on x through the parameters of the Frechet distribution
y <- rfrechet(n,loc = 3*x+1,scale = 0.5*x+0.001,shape=x+1)

plot(x,y,pch=22)
# 0.99 quantile of the generative distribution
qgen <- qfrechet(0.99,loc = 3*x+1,scale = 0.5*x+0.001,shape=x+1)
points(x,qgen,pch="*",col="orange")

h <- 2 # number of hidden units
m <- 4 # number of components

# initialize a conditional mixture with hybrid Pareto components
thetainit <- condhparetomixt.init(1,h,m,y)

# MLE for initial neural network parameters
condhparetomixt.fit(thetainit,h,m,t(x),y,iterlim=100)
```

condmixt.foldtrain *Training of conditional mixtures and evaluation of the negative log-likelihood on validation data*

Description

This function can be used to parallelize n-fold cross-validation. Training is done on a training set and the negative log-likelihood is evaluated on validation data. This is repeated for a set of hyper-parameter values. Model selection can be performed based on the evaluation of each set of hyper-parameters on validation data.

Usage

```
condhparetomixt.foldtrain(xtrain, ytrain, xtest, ytest, hp, nstart = 1, ...)
condhparetomixt.foldtrain.tailpen(xtrain, ytrain, xtest, ytest, hp, nstart = 1, ...)
condhparetomixt.dirac.foldtrain.tailpen(xtrain, ytrain, xtest, ytest, hp, nstart = 1, ...)
condgaussmixt.foldtrain(xtrain, ytrain, xtest, ytest, hp, nstart = 1, ...)
condgaussmixt.dirac.foldtrain(xtrain, ytrain, xtest, ytest, hp, nstart = 1, ...)
condlognormixt.foldtrain(xtrain, ytrain, xtest, ytest, hp, nstart = 1, ...)
condlognormixt.dirac.foldtrain(xtrain, ytrain, xtest, ytest, hp, nstart = 1, ...)
condbergamixt.foldtrain(xtrain, ytrain, xtest, ytest, hp,nstart = 1, ...)
```

Arguments

xtrain	Matrix of explanatory (independent) variables of dimension d x ntrain, d is the number of variables and ntrain is the number of training examples (patterns)
ytrain	Vector of ntrain dependent variables
xtest	Matrix of explanatory (independent) variables of dimension d x ntest, d is the number of variables and ntest is the number of test/validation examples (patterns)
ytest	Vector of ntest dependent variables
hp	Matrix of hyper-parameters where the columns represent the different hyper-parameters and the rows the sets of values for each hyper-parameter. For condhparetomixt, condgaussmixt and lognormixt together with the version with a dirac component, the hyper-parameters are the number of hidden units and the number of components. When a tail penalty is included for the hybrid Pareto conditional mixture, there are possibly five other hyper-parameters: lambda, w, beta, mu and sigma which controls the tail penalty, see in the Details section below.
nstart	Number of minimization re-starts, default is one.
...	optional arguments for the optimizer nlm

Details

condhparetomixt indicates a mixture with hybrid Pareto components, condgaussmixt for Gaussian components, condlognormixt for Log-Normal components, condbergam for a Bernoulli-Gamma two component mixture, tailpen indicates that a penalty is added to the log-likelihood

to guide the tail index parameter estimation, `dirac` indicates that a discrete dirac component at zero is included in the mixture

In order to drive the tail index estimation, a penalty is introduced in the log-likelihood. The goal of the penalty is to include a priori information which in our case is that only a few mixture components have a heavy tail index which should approximate the tail of the underlying distribution while most other mixture components have a light tail and aim at modelling the central part of the underlying distribution.

The penalty term is given by the logarithm of the following two-component mixture, as a function of a tail index parameter xi :

w beta exp(-beta xi) + (1-w) exp(-(xi-mu)^2/(2 sigma^2))/(sqrt(2 pi) sigma)

where the first term is the prior on the light tail component and the second term is the prior on the heavy tail component.

The extra hyper-parameters for the penalty terms are as follows : - lambda : penalty parameter which controls the trade-off between the penalty and the negative log-likelihood, takes on positive values. If zero, no penalty - w : penalty parameter in [0,1] which is the proportion of components with light tails, 1-`w` being the proportion of components with heavy tails - beta : positive penalty parameter which indicates the importance of the light tail components (it is the parameter of an exponential which represents the prior over the light tail components) - mu : penalty parameter in (0,1) which represents the a priori value for the heavy tail index of the underlying distribution - sigma : positive penalty parameter which controls the spread around the a priori value for the heavy tail index of the underlying distribution

Value

Returns a vector of negative log-likelihood values evaluated on the test set corresponding to each set of hyper-parameters.

Author(s)

Julie Carreau

References

Bishop, C. (1995), Neural Networks for Pattern Recognition, Oxford

Carreau, J. and Bengio, Y. (2009), A Hybrid Pareto Mixture for Conditional Asymmetric Fat-Tailed Distributions, 20, IEEE Transactions on Neural Networks

See Also

`condmixt.train`,`condmixt.nll`, `condmixt.init`

Examples

```
# generate train data
ntrain <- 200
xtrain <- runif(ntrain)
ytrain <- rfrechet(ntrain,loc = 3*xtrain+1,scale =
0.5*xtrain+0.001,shape=xtrain+1)
```

```
plot(xtrain,ytrain,pch=22) # plot train data
qgen <- qfrechet(0.99,loc = 3*xtrain+1,scale = 0.5*xtrain+0.001,shape=xtrain+1)
points(xtrain,qgen,pch="*",col="orange")

# generate test data
ntest <- 200
xtest <- runif(ntest)
ytest <- rfrechet(ntest,loc = 3*xtest+1,scale =
0.5*xtest+0.001,shape=xtest+1)

# create a matrix with sets of values for the number of hidden units and
# the number of components
hp <- matrix(c(2,4,2,2),nrow=2,ncol=2)

# train and test a mixture with hybrid Pareto components
condhparetomixt.foldtrain(t(xtrain),ytrain,t(xtest),ytest,hp,nstart=2,iterlim=100)
```

`condmixt.fwd` *Forward pass in neural network conditional mixtures*

Description

A forward pass means that given explanatory variables `x`, the neural network computes the corresponding values of the mixture parameters.

Usage

```
condhparetomixt.fwd(theta, h, m, x)
condhparetomixt.dirac.fwd(theta, h, m, x)
condgaussmixt.fwd(theta, h, m, x)
condgaussmixt.dirac.fwd(theta, h, m, x)
condbergamixt.fwd(theta,h,x)
```

Arguments

`theta`	Vector of neural network parameters
`h`	Number of hidden units
`m`	Number of components
`x`	Matrix of explanatory (independent) variables of dimension d x n, d is the number of variables and n is the number of examples (patterns)

Details

`condhparetomixt` indicates a mixture with hybrid Pareto components, `condgaussmixt` for Gaussian components, `condbergam` for a Bernoulli-Gamma two component mixture, `dirac` indicates that a discrete dirac component is included in the mixture The forward pass for Log-Normal conditional mixture is the same one as for Gaussian conditional mixture. Therefore, `condgaussmixt.fwd` can be used for the forward pass of a conditional mixture with Log-Normal components.

Value

A matrix of mixture parameters corresponding to the values in x : - for condhparetomixt.fwd, each component requires four parameters (pi, xi, mu, sigma) and the parameter matrix has dimensions m x 4 x n - for condhparetomixt.dirac.fwd, there is an additional parameter for the probability of the dirac at zero so that the mixture parameters are stored in a (4m+1) x n matrix - for condgaussmixt.fwd, each component requires three parameters (pi, mu, sigma) and the parameter matrix has dimensions m x 3 x n - for condgaussmixt.dirac.fwd, there is an additional parameter for the probability of the dirac at zero so that the mixture parameters are stored in a (3m+1) x n matrix - for condbergamixt.fwd, there are three parameters, the probability of the dirac at zero and two parameters for the Gamma distribution

Author(s)

Julie Carreau

References

Bishop, C. (1995), Neural Networks for Pattern Recognition, Oxford

Carreau J. and Vrac, M. (2011) Stochastic Downscaling of Precipitation with Neural Network Conditional Mixture Models, 47, Water Resources Research

Williams, M.P. (1998) Modelling Seasonality and Trends in Daily Rainfall Data, 10, Advances in Neural Information and Processing Systems

See Also

condmixt condmixt.init, condmixt.nll

Examples

```
n <- 200
x <- runif(n) # x is a random uniform variate
# y depends on x through the parameters of the Frechet distribution
y <- rfrechet(n,loc = 3*x+1,scale = 0.5*x+0.001,shape=x+1)
plot(x,y,pch=22)
# 0.99 quantile of the generative distribution
qgen <- qfrechet(0.99,loc = 3*x+1,scale = 0.5*x+0.001,shape=x+1)
points(x,qgen,pch="*",col="orange")

h <- 2 # number of hidden units
m <- 4 # number of components

# initialize a conditional mixture with hybrid Pareto components
thetainit <- condhparetomixt.init(1,h,m,y)

params.mixt <- condhparetomixt.fwd(thetainit,h,m,t(x)) # compute mixture parameters
```

condmixt.init *Conditional mixture parameter initial values*

Description

Neural network weights are randomly initialized uniformly over the range [-0.9/sqrt(k),0.9/sqrt(k)] where k is the number of inputs to the neuron. This ensures that the hidden units will not be saturated and that training should proceed properly. In addition, if the dependent data Y is provided, the biases will be initialized according to the initial parameters of an unconditional mixture computed on the dependent data.

Usage

```
condhparetomixt.init(d, h, m, y = NULL)
condhparetomixt.dirac.init(d, h, m, y = NULL)
condgaussmixt.init(d,h,m,y=NULL)
condgaussmixt.dirac.init(d,h,m,y=NULL)
condlognormixt.init(d,h,m,y=NULL)
condlognormixt.dirac.init(d,h,m,y=NULL)
condbergamixt.init(d,h,y=NULL)
```

Arguments

d	dimension of input x to neural network
h	number of hidden unit
m	number of components
y	optional, dependent one-dimensional data

Details

If the argument y is provided, an unconditional mixture with the same type of components will be initialized on y. These initial unconditional parameters are then used to give more appropriate initial values to the biases of the neural network.

Value

A vector of neural network parameters for the given number of hidden units and number of components and specific conditional mixture formulation : hybrid Pareto components (condhparetomixt.init), hybrid Pareto components + discrete dirac component at zero (condhparetomixt.dirac.init), Gaussian components (condgaussmixt.init), Gaussian components + discrete dirac component at zero (condgaussmixt.dirac.init), Log-Normal components (condlognormixt.init), Log-Normal components + discrete dirac component at zero (condlognormixt.dirac.init) and the Bernoulli-Gamma two-component mixture (condbergamixt.init).

Author(s)

Julie Carreau

References

Carreau, J. and Bengio, Y. (2009), A Hybrid Pareto Mixture for Conditional Asymmetric Fat-Tailed Distributions, 20, IEEE Transactions on Neural Networks

Nabney, I. (2002) NetLab : Algorithms for Pattern Recognition, Springer

Williams, M.P. (1998) Modelling Seasonality and Trends in Daily Rainfall Data, 10, Advances in Neural Information and Processing Systems

See Also

hparetomixt.init, gaussmixt.init

Examples

```
n <- 200
x <- runif(n) # x is a random uniform variate
# y depends on x through the parameters of the Frechet distribution
y <- rfrechet(n,loc = 3*x+1,scale = 0.5*x+0.001,shape=x+1)

plot(x,y,pch=22)
# 0.99 quantile of the generative distribution
qgen <- qfrechet(0.99,loc = 3*x+1,scale = 0.5*x+0.001,shape=x+1)
points(x,qgen,pch="*",col="orange")

h <- 2 # number of hidden units
m <- 4 # number of components

# initialize a conditional mixture with hybrid Pareto components
thetainit <- condhparetomixt.init(1,h,m,y)
```

---

condmixt.nll *Negative log-likelihood for conditional mixtures*

Description

Computes negative log-likelihood and gradient for given neural network parameters, numbers of hidden units and of components, the type of components and the presence of a discrete dirac component on a given data set.

Usage

```
condhparetomixt.nll(theta, h, m, x, y)
condhparetomixt.nll.tailpen(theta,h,m,x,y,lambda=0,w=0.2,beta=50,mu=0.2,sigma=0.2)
condhparetomixt.dirac.nll.tailpen(theta,h,m,x,y,lambda=0,w=0.2,beta=50,mu=0.2,sigma=0.2)
condhparetomixt.dirac.nll(theta,h,m,x,y)
condgaussmixt.nll(theta,h,m,x,y)
condgaussmixt.dirac.nll(theta,h,m,x,y)
condlognormixt.nll(theta,h,m,x,y)
condlognormixt.dirac.nll(theta,h,m,x,y)
condbergamixt.nll(theta,h,x,y)
```

Arguments

theta	Vector of neural network parameters
h	Number of hidden units
m	Number of components
x	Matrix of explanatory (independent) variables of dimension d x n, d is the number of variables and n is the number of examples (patterns)
y	Vector of n dependent variables
lambda	penalty parameter which controls the trade-off between the penalty and the negative log-likelihood, takes on positive values. If zero, no penalty
w	penalty parameter in [0,1] which is the proportion of components with light tails, 1-w being the proportion of components with heavy tails
beta	positive penalty parameter which indicates the importance of the light tail components (it is the parameter of an exponential which represents the prior over the light tail components)
mu	penalty parameter in (0,1) which represents the a priori value for the heavy tail index of the underlying distribution
sigma	positive penalty parameter which controls the spread around the a priori value for the heavy tail index of the underlying distribution

Details

condhparetomixt indicates a mixture with hybrid Pareto components, condgaussmixt for Gaussian components, condlognormixt for Log-Normal components, condbergam for a Bernoulli-Gamma two component mixture, tailpen indicates that a penalty is added to the log-likelihood to guide the tail index parameter estimation, dirac indicates that a discrete dirac component at zero is included in the mixture

In order to drive the tail index estimation, a penalty is introduced in the log-likelihood. The goal of the penalty is to include a priori information which in our case is that only a few mixture components have a heavy tail index which should approximate the tail of the underlying distribution while most other mixture components have a light tail and aim at modelling the central part of the underlying distribution.

The penalty term is given by the logarithm of the following two-component mixture, as a function of a tail index parameter xi :

w beta exp(-beta xi) + (1-w) exp(-(xi-mu)^2/(2 sigma^2))/(sqrt(2 pi) sigma)

where the first term is the prior on the light tail component and the second term is the prior on the heavy tail component.

Value

Returns a single value (the negative log-likelihood for given parameters and sample) and a vector, the gradient, which is passed as an attribute.

Author(s)

Julie Carreau

References

Bishop, C. (1995), Neural Networks for Pattern Recognition, Oxford

Carreau, J. and Bengio, Y. (2009), A Hybrid Pareto Mixture for Conditional Asymmetric Fat-Tailed Distributions, 20, IEEE Transactions on Neural Networks

See Also

hparetomixt.negloglike, hparetomixt.negloglike.tailpen, condmixt.init

Examples

```
n <- 200
x <- runif(n) # x is a random uniform variate
# y depends on x through the parameters of the Frechet distribution
y <- rfrechet(n,loc = 3*x+1,scale = 0.5*x+0.001,shape=x+1)
plot(x,y,pch=22)
# 0.99 quantile of the generative distribution
qgen <- qfrechet(0.99,loc = 3*x+1,scale = 0.5*x+0.001,shape=x+1)
points(x,qgen,pch="*",col="orange")

h <- 2 # number of hidden units
m <- 4 # number of components

# initialize a conditional mixture with hybrid Pareto components
thetainit <- condhparetomixt.init(1,h,m,y)

# computes negative log-likelihood and gradient for initial neural network parameters
condhparetomixt.nll(thetainit,h,m,t(x),y)
```

---

condmixt.quant *Quantile computation for conditional mixtures.*

Description

Quantile computation for conditional mixtures requires to solve numerically F(y)=p where F is the distribution function of the conditional mixture and p is a probability level.

Usage

```
condhparetomixt.quant(theta, h, m, x, p, a, b, trunc = TRUE)
condhparetomixt.dirac.quant(theta,h,m,x,p,a,b)
condhparetomixt.dirac.condquant(theta,h,m,x,p,a,b)
condgaussmixt.quant(theta,h,m,x,p,a,b,trunc=TRUE)
condgaussmixt.dirac.quant(theta,h,m,x,p,a,b)
condgaussmixt.dirac.condquant(theta,h,m,x,p,a,b)
condlognormixt.quant(theta,h,m,x,p,a,b)
condlognormixt.dirac.quant(theta,h,m,x,p,a,b)
```

```
condlognormixt.dirac.condquant(theta,h,m,x,p,a,b)
condbergamixt.quant(theta,h,x,p)
```

Arguments

theta	Vector of neural network parameters
h	Number of hidden units
m	Number of components
x	Matrix of explanatory (independent) variables of dimension d x n, d is the number of variables and n is the number of examples (patterns)
p	Probability level in [0,1]
a	Approximate lower bound on quantile value.
b	Approximate upper bound on quantile value.
trunc	Logical variable, if true, density is truncated below zero and re-weighted to make sure it integrates to one.

Details

condhparetomixt indicates a mixture with hybrid Pareto components, condgaussmixt for Gaussian components, condlognormixt for Log-Normal components, condbergam for a Bernoulli-Gamma two component mixture, dirac indicates that a discrete dirac component is included in the mixture condquant applies for mixtures with a dirac component at zero : quantiles are computed given that the variable is strictly positive, that is the quantile is computed for the continuous part of the mixture only : P(Y <= y | Y >0, X)

Value

Computed quantiles are stored in a matrix whose rows correspond to the probability levels and whose columns correspond to the number of examples n.

Author(s)

Julie Carreau

References

Bishop, C. (1995), Neural Networks for Pattern Recognition, Oxford

Carreau, J. and Bengio, Y. (2009), A Hybrid Pareto Mixture for Conditional Asymmetric Fat-Tailed Distributions, 20, IEEE Transactions on Neural Networks

See Also

condmixt.train,condmixt.nll, condmixt.init

Examples

```
# generate train data
ntrain <- 200
xtrain <- runif(ntrain)
ytrain <- rfrechet(ntrain,loc = 3*xtrain+1,scale =
0.5*xtrain+0.001,shape=xtrain+2)
plot(xtrain,ytrain,pch=22) # plot train data
qgen <- qfrechet(0.99,loc = 3*xtrain+1,scale = 0.5*xtrain+0.001,shape=xtrain+2)
points(xtrain,qgen,pch="*",col="orange")

# generate test data
ntest <- 200
xtest <- runif(ntest)
ytest <- rfrechet(ntest,loc = 3*xtest+1,scale =
0.5*xtest+0.001,shape=xtest+2)

h <- 2 # number of hidden units
m <- 4 # number of components

# train a mixture with hybrid Pareto components
thetaopt <- condhparetomixt.train(h,m,t(xtrain),ytrain, nstart=2,iterlim=100)
qmod <- condhparetomixt.quant(thetaopt,h,m,t(xtest),0.99,0,10,trunc=TRUE)
points(xtest,qmod,pch="o",col="blue")
```

`condmixt.train` *Training of conditional mixtures*

Description

Training involves, for given numbers of hidden units and components and a given mixture specification, minimizing the negative log-likelihood from initial parameter values. The minimization is re-started several times from various initial parameter values and the best minimum is kept. This helps avoiding local minima.

Usage

```
condhparetomixt.train(h, m, x, y, nstart = 1, ...)
condhparetomixt.train.tailpen(h,m,x,y,lambda=0,w=0.2,beta=50,
                              mu=0.2,sigma=0.2, nstart = 1, ...)
condhparetomixt.dirac.train.tailpen(h,m,x,y,lambda=0,w=0.2,beta=50,
                                    mu=0.2,sigma=0.2, nstart = 1, ...)
condgaussmixt.train(h,m,x,y, nstart = 1, ...)
condgaussmixt.dirac.train(h,m,x,y, nstart = 1, ...)
condlognormixt.train(h,m,x,y, nstart = 1, ...)
condlognormixt.dirac.train(h,m,x,y, nstart = 1, ...)
condbergamixt.train(h,x,y, nstart = 1, ...)
```

Arguments

h	Number of hidden units
m	Number of components
x	Matrix of explanatory (independent) variables of dimension d x n, d is the number of variables and n is the number of examples (patterns)
y	Vector of n dependent variables
nstart	Number of minimization re-starts, default is one.
lambda	penalty parameter which controls the trade-off between the penalty and the negative log-likelihood, takes on positive values. If zero, no penalty
w	penalty parameter in [0,1] which is the proportion of components with light tails, 1-w being the proportion of components with heavy tails
beta	positive penalty parameter which indicates the importance of the light tail components (it is the parameter of an exponential which represents the prior over the light tail components)
mu	penalty parameter in (0,1) which represents the a priori value for the heavy tail index of the underlying distribution
sigma	positive penalty parameter which controls the spread around the a priori value for the heavy tail index of the underlying distribution
...	optional arguments for the optimizer nlm

Details

condhparetomixt indicates a mixture with hybrid Pareto components, condgaussmixt for Gaussian components, condlognormixt for Log-Normal components, condbergam for a Bernoulli-Gamma two component mixture, tailpen indicates that a penalty is added to the log-likelihood to guide the tail index parameter estimation, dirac indicates that a discrete dirac component at zero is included in the mixture

In order to drive the tail index estimation, a penalty is introduced in the log-likelihood. The goal of the penalty is to include a priori information which in our case is that only a few mixture components have a heavy tail index which should approximate the tail of the underlying distribution while most other mixture components have a light tail and aim at modelling the central part of the underlying distribution.

The penalty term is given by the logarithm of the following two-component mixture, as a function of a tail index parameter xi :

w beta exp(-beta xi) + (1-w) exp(-(xi-mu)^2/(2 sigma^2))/(sqrt(2 pi) sigma)

where the first term is the prior on the light tail component and the second term is the prior on the heavy tail component.

Value

Returns a vector of neural network parameters corresponding to the best minimum among nstart optimizations.

Author(s)

Julie Carreau

References

Bishop, C. (1995), Neural Networks for Pattern Recognition, Oxford

Carreau, J. and Bengio, Y. (2009), A Hybrid Pareto Mixture for Conditional Asymmetric Fat-Tailed Distributions, 20, IEEE Transactions on Neural Networks

See Also

condmixt.train,condmixt.nll, condmixt.init

Examples

```
n <- 200
x <- runif(n) # x is a random uniform variate
# y depends on x through the parameters of the Frechet distribution
y <- rfrechet(n,loc = 3*x+1,scale = 0.5*x+0.001,shape=x+1)
plot(x,y,pch=22)
# 0.99 quantile of the generative distribution
qgen <- qfrechet(0.99,loc = 3*x+1,scale = 0.5*x+0.001,shape=x+1)
points(x,qgen,pch="*",col="orange")

h <- 2 # number of hidden units
m <- 4 # number of components

# train a mixture with hybrid Pareto components
condhparetomixt.train(h,m,t(x),y,nstart=2,iterlim=100)
```

---

gaussmixt *Mixture of Gaussians*

Description

Density and distribution function for a mixture of Gaussians with m components.

Usage

```
dgaussmixt(params, x, log = FALSE, trunc = TRUE)
pgaussmixt(params, x, trunc = TRUE)
```

Arguments

params	matrix of mixture parameters of dimension 3 x m, where m is the number of components, so that each column contains the mixture parameters (pi, mu, sigma) related to one component
x	vector of sample data

log	logical, if TRUE, probabilities p are given as log(p).
trunc	logical, if TRUE (default), the Gaussian density is truncated below zero.

Value

dgaussmixt gives the density and pgaussmixt gives the distribution function

Author(s)

Julie Carreau

See Also

Normal

---

gaussmixt.init *Provides initial values for the parameters of a mixture of Gaussians based on a sample.*

Description

Initial values for the parameters of a mixture of Gaussians are provided by applying the following steps : 1) clustering the sample into as many clusters as there are mixture components 2) the initial means and standard deviations of each component are taken as the cluster centers and median absolute deviation respectively computed on each component

Usage

```
gaussmixt.init(m, x, iter.max = 20, nstart = 10)
```

Arguments

m	number of mixture components
x	data sample from which the initial parameters are computed
iter.max	maximum number of iteration for kmeans clustering, default is 20, see kmeans
nstart	number of random cluster centers chosen (default is 10), see kmeans

Value

a matrix of dimension 3 x m which stores the 3 parameters (pi, mu, sigma) of each of the m components.

Author(s)

Julie Carreau

References

McLachlan, G. and Peel, D. (2000), Finite Mixture Models, Wiley series in probability and statistics

See Also

kmeans,hparetomixt.init

Examples

```
r <- rfrechet(500,loc=5,scale=5,shape=10)
m <- 2
param.init <- gaussmixt.init(2,r)
```

---

gpd.mme — *Moment Estimator for the Generalized and the Hybrid Pareto Distribution*

Description

Moment estimators for the generalized Pareto distribution and parameter estimators based on two quantiles plus a tail index estimator for the hybrid Pareto distribution.

Usage

```
gpd.mme(x)
hpareto.mme(x, xi0=c(),p=0.99)
```

Arguments

x	vector of sample for which the parameters will be estimated
xi0	an optional a priori value for the tail index parameter
p	the percentage of largest observations used for tail index estimation

Details

For hpareto.mme, the tail index is assumed to be positive. In case one has some prior information on the value of the tail index parameter, it is possible to provide this value as an argument to the function hpareto.mme. The two other parameters mu and sigma will be estimated based on that tail index estimate and two quantiles of the hybrid Pareto distribution. If the tail index parameter is not provided as input, it will be estimated with the Hill estimator using data above the p-quantile. By default, p=0.99 but this might be inappropriate depending on the sample. Since the tail index estimate is very sensitive, it is recommended to tune carefully the p argument.

Value

A vector of parameter estimates.

Author(s)

Julie Carreau

References

Hosking, J. R. M. and Wallis, J. R. (1987), Parameter and quantile estimation for the Generalized Pareto distribution, 29, Technometrics Carreau, J. and Bengio, Y. (2009), A Hybrid Pareto Model for Asymmetric Fat-tailed Data: the Univariate Case, 12, Extremes

Examples

```
r<-rhpareto(1000,0.1,0,1,trunc=FALSE)
hpareto.mme(r,p=0.991)
```

---

hillest *Hill Estimator*

Description

Hill estimator of the tail index. This estimator assumes the tail index to be positive. The threshold used is the k+1 order statistic.

Usage

```
hillest(data, k)
```

Arguments

data	vector of sample from a distribution for which the tail index is to be estimated
k	define the number of order statistics used for the estimation

Value

Hill estimator of the tail index parameter.

Author(s)

Julie Carreau

References

Embrechts, P., Kluppelberg, C. and Mikosch, T. (1997), Modelling Extremal Events, Applications of Mathematics, Stochastic Modelling and Applied Probability, Springer

See Also

gpd.mme, hpareto.mme

Examples

```
r<-rhpareto(5000,0.2,-50,1,trunc=TRUE)  # tail index is equal to 0.2
 # Hill estimator of the tail index with the 100 largest observations
hillest(r,10)
```

---

hpareto *The Hybrid Pareto Distribution*

Description

Density, distribution function, quantile function and random generation for the hybrid Pareto distribution with parameters xi, mu and sigma.

Usage

```
dhpareto(y, xi, mu = 0, sigma = 1, log = FALSE, trunc = TRUE)
phpareto(q, xi, mu = 0, sigma = 1, trunc = TRUE)
qhpareto(p, xi, mu = 0, sigma = 1, trunc = TRUE)
rhpareto(n, xi, mu = 0, sigma = 1, trunc = TRUE)
```

Arguments

y,q	vector of quantiles
p	vector of probabilities
n	number of observations
xi	tail index parameter, inherited from the GPD
mu	location parameter, inherited from the Gaussian
sigma	scale parameter, inheristed from the Gaussian
log	logical, if TRUE, probabilities p are given as log(p).
trunc	logical, if TRUE (default), the hybrid Pareto density is truncated below zero.

Details

The hybrid Pareto density is given by a Gaussian with parameters mu and sigma below the threshold alpha (see the function hpareto.alpha) and by the GPD with parameters xi and beta.(see the function hpareto.beta) To ensure continuity of the density and of its derivative at the threshold, alpha and beta are appropriate functions of xi, mu and sigma. Appropriate reweighting factor gamma ensures that the density integrate to one.

Value

dhpareto gives the density, phpareto gives the distribution function, qhpareto gives the quantile function and rhpareto generates random deviates.

Author(s)

Julie Carreau

References

Carreau, J. and Bengio, Y. (2009), A Hybrid Pareto Model for Asymmetric Fat-tailed Data: the Univariate Case, 12, Extremes

See Also

hpareto.alpha, hpareto.beta and hpareto.gamma

---

hpareto.alpha *Auxillary Parameters of the Hybrid Pareto Distribution*

Description

Computes the junction point alpha, the GPD scale parameter beta and the normalization factor gamma of the hybrid Pareto distributions.

Usage

```
hpareto.alpha(xi, mu = 0, sigma = 1)
hpareto.beta(xi, sigma = 1)
hpareto.gamma(xi, mu = 0, sigma = 1, trunc = T)
```

Arguments

xi	Tail index parameter of the hybrid Pareto distribution.
mu	Location parameter of the hybrid Pareto distribution.
sigma	Scale parameter of the hybrid Pareto distribution.
trunc	Binary variable : if TRUE, the density of the hybrid Pareto is truncated below zero

Details

Let z = (1+xi)^2/(2 pi) and W be the Lambert W function implemented in lambertw. Then : alpha = mu + sigma * sqrt(W(z)), beta = sigma * |1 + xi| / sqrt(W(z)) and gamma is the integral from 0 (is trunc is true, -infinity otherwise) to alpha.

Value

Computation of the auxillary parameters alpha, beta and gamma.

Author(s)

Julie Carreau

References

Carreau, J. and Bengio, Y. (2009), A Hybrid Pareto Model for Asymmetric Fat-tailed Data: the Univariate Case, 12, Extremes

See Also

dhpareto,phpareto and rhpareto

Examples

```
hpareto.alpha(0.1,0,1)
hpareto.beta(0.1,1)
hpareto.gamma(0.1,0,1)
```

---

hpareto.negloglike *Hybrid Pareto Maximum Likelihood Estimation*

Description

Negative log-likelihood and gradient (hpareto.negloglike) and MLE of a hybrid Pareto distribution parameters (hpareto.fit). hpareto.fit applies the optimizer nlm to minimize the negative log-likelihood based on some starting values for the hybrid Pareto parameters.

Usage

```
hpareto.negloglike(params, x)
hpareto.fit(params, x, ...)
```

Arguments

params	vector of length 3 of hybrid Pareto parameters c(xi, mu, sigma)
x	a vector of length n of observations assumed to be sampled from a hybrid Pareto distribution
...	optional arguments for nlm called by hpareto.fit

Value

hpareto.negloglike returns a single value (the negative log-likelihood for given parameters and sample) and a vector, the gradient, which is passed as an attribute, while hpareto.fit returns a vector of three parameters, the MLE of the parameters of the hybrid Pareto distribution

Author(s)

Julie Carreau

References

Carreau, J. and Bengio, Y. (2009), A Hybrid Pareto Model for Asymmetric Fat-tailed Data: the Univariate Case, 12, Extremes

See Also

hpareto.mme, dhpareto,phpareto and rhpareto

Examples

```
r <- rhpareto(500,0.2,trunc=FALSE)
params.init <- hpareto.mme(r)
hpareto.negloglike(params.init,r)
hpareto.fit(params.init,r)
```

---

hparetomixt *Mixture of hybrid Paretos*

Description

Density and distribution function for a mixture of hybrid Paretos with m components.

Usage

```
dhparetomixt(params, x, log = FALSE, trunc = TRUE)
phparetomixt(params, x, trunc = TRUE)
```

Arguments

params	matrix of mixture parameters of dimension 4 x m, where m is the number of components, so that each column contains the mixture parameters (pi, xi, mu, sigma) related to one component
x	vector of sample data
log	logical, if TRUE, probabilities p are given as log(p).
trunc	logical, if TRUE (default), the hybrid Pareto density is truncated below zero.

Value

dhparetomixt gives the density and phparetomixt gives the distribution function

Author(s)

Julie Carreau

References

Carreau, J. and Bengio, Y. (2009), A Hybrid Pareto Model for Asymmetric Fat-tailed Data: the Univariate Case, 12, Extremes

See Also

hparetomixt.init

---

hparetomixt.disp *Display the Hybrid Pareto Mixture Parameters*

Description

Display the Hybrid Pareto Mixture Parameters in Latex Style Tabular form

Usage

```
hparetomixt.disp(params)
```

Arguments

params	matrix of mixture parameters of dimension 4 x m, where m is the number of components, so that each column contains the mixture parameters (pi, xi, mu, sigma) related to one component

Value

Display on the R consol the hybrid Pareto mixture parameters with syntax suitable to make a Latex tabular out of it.

Author(s)

Julie Carreau

See Also

hparetomixt, hparetomixt.negloglike

Examples

```
r <- rfrechet(500,loc=5,scale=5,shape=5)
m <- 2
param.init <- hparetomixt.init(m,r)
hparetomixt.disp(param.init)
```

`hparetomixt.init` *Provides initial values for the parameters of a mixture of hybrid Paretos based on a sample.*

Description

Initial values for the parameters of a mixture of hybrid Paretos are provided by applying the following steps : 1) clustering the sample into as many clusters as there are mixture components 2) estimating the hybrid Pareto parameters for each component on the data from each cluster with the moment-like estimators, see `hpareto.mme`

Usage

```
hparetomixt.init(m, x, iter.max = 20, nstart = 10)
```

Arguments

`m`	number of mixture components
`x`	data sample from which the initial parameters are computed
`iter.max`	maximum number of iteration for kmeans clustering, default is 20, see `kmeans`
`nstart`	number of random cluster centers chosen (default is 10), see `kmeans`

Value

a matrix of dimension 4 x `m` which stores the 4 parameters (pi, xi, mu, sigma) of each of the `m` components.

Author(s)

Julie Carreau

References

Carreau, J. and Bengio, Y. (2009), A Hybrid Pareto Model for Asymmetric Fat-tailed Data: the Univariate Case, 12, Extremes

See Also

`kmeans`, `hpareto.mme`

Examples

```
r <- rfrechet(500,loc=5,scale=5,shape=5)
m <- 2
param.init <- hparetomixt.init(m,r)
```

`hparetomixt.negloglike` *Maximum Likelihood Estimation for a Mixture of Hybrid Paretos*

Description

Negative log-likelihood and gradient (`hparetomixt.negloglike`), MLE of a hybrid Pareto distribution parameters (`hparetomixt.fit`) and out-of-sample negative log-likelihood estimation for a given number of components with nfold cross-validation (`hparetomixt.cvtrain`).

`hparetomixt.fit` applies the optimizer `nlm` to minimize the negative log-likelihood based on some starting values for the hybrid Pareto parameters.

Usage

```
hparetomixt.negloglike(params, x)
hparetomixt.fit(params, x, ...)
hparetomixt.cvtrain(m, x, nfold=5, nstart=1, ...)
```

Arguments

`params`	matrix of dimension 4 by m, where m is the number of components, each column of the matrix contains the mixture parameters of one component (pi, xi, mu, sigma)
`x`	a vector of length n of observations assumed to be sampled from a mixture of hybrid Paretos
`m`	number of mixture components
`nfold`	number of fold for cross-validation estimate, default is 5
`nstart`	number of re-starts for the optimizer `nlm` with different initial parameters, default is 1
`...`	optional arguments for `nlm`

Value

`hparetomixt.negloglike` returns a single value (the negative log-likelihood for given parameters and sample) and a vector, the gradient, which is passed as an attribute, while `hparetomixt.fit` returns a 4 by m matrix of MLE for the hybrid Pareto mixture parameters and `hparetomixt.cvtrain` returns a cross-validation estimate of the out-of-sample negative log-likelihood for a selected number of components

Author(s)

Julie Carreau

References

Carreau, J. and Bengio, Y. (2009), A Hybrid Pareto Model for Asymmetric Fat-tailed Data: the Univariate Case, 12, Extremes

See Also

hparetomixt.init, hparetomixt.negloglike.tailpen

Examples

```
r <- rfrechet(500,loc=5,scale=5,shape=5)
m <- 2
param.init <- hparetomixt.init(m,r)
hparetomixt.negloglike(param.init,r)
hparetomixt.fit(param.init,r)
```

---

hparetomixt.negloglike.tailpen

Maximum Likelihood Estimation for a Mixture of Hybrid Paretos with Tail Penalty

---

Description

In order to drive the tail index estimation, a penalty is introduced in the log-likelihood. The goal of the penalty is to include a priori information which in our case is that only a few mixture components have a heavy tail index which should approximate the tail of the underlying distribution while most other mixture components have a light tail and aim at modelling the central part of the underlying distribution.

Usage

```
hparetomixt.negloglike.tailpen(params, lambda, w, beta, mu, sigma, x)
hparetomixt.fit.tailpen(params, lambda, w, beta, mu, sigma, x, ...)
hparetomixt.cvtrain.tailpen(m, lambda, w, beta, mu, sigma, x, nfold=5, nstart=1, ...)
```

Arguments

params	matrix of dimension 4 by m, where m is the number of components, each column of the matrix contains the mixture parameters of one component (pi, xi, mu, sigma)
m	number of mixture components
lambda	penalty parameter which controls the trade-off between the penalty and the negative log-likelihood, takes on positive values
w	penalty parameter in [0,1] which is the proportion of components with light tails, 1-w being the proportion of components with heavy tails
beta	positive penalty parameter which indicates the importance of the light tail components (it is the parameter of an exponential which represents the prior over the light tail components)
mu	penalty parameter in (0,1) which represents the a priori value for the heavy tail index of the underlying distribution

sigma	positive penalty parameter which controls the spread around the a priori value for the heavy tail index of the underlying distribution
x	a vector of length n of observations assumed to be sampled from a mixture of hybrid Paretos
nfold	number of fold for cross-validation estimate, default is 5
nstart	number of re-starts for the optimizer nlm with different initial parameters, default is 1
...	optional arguments for nlm

Details

The penalty term is given by the logarithm of the following two-component mixture, as a function of a tail index parameter xi : w beta exp(-beta xi) + (1-w) exp(-(xi-mu)^2/(2 sigma^2))/(sqrt(2 pi) sigma) where the first term is the prior on the light tail component and the second term is the prior on the heavy tail component.

Value

hparetomixt.negloglike.tailpen returns a single value (the negative log-likelihood for given parameters and sample) and a vector, the gradient, which is passed as an attribute, while hparetomixt.fit.tailpen returns a 4 by m matrix of MLE for the hybrid Pareto mixture parameters and hparetomixt.cvtrain.tailpen returns a cross-validation estimate of the out-of-sample negative log-likelihood for the given model (number of components and penalty parameters)

Author(s)

Julie Carreau

References

Carreau, J.,Naveau, P. and Sauquet, E. (2009), A statistical rainfall-runoff mixture model with heavy-tailed components, 45, Water Resources Research

See Also

hparetomixt.init, hparetomixt.negloglike

Examples

```
r <- rfrechet(500,loc=5,scale=5,shape=5)
m <- 2
param.init <- hparetomixt.init(m,r)
hparetomixt.negloglike.tailpen(param.init,10,0.5,20,0.1,0.2,r)
hparetomixt.fit.tailpen(param.init,10,0.5,20,0.1,0.2,r)
hparetomixt.cvtrain.tailpen(2,10,0.5,20,0.1,0.2,r)
```

`kneigh.condquant` *Conditional quantile estimation from nearest neighbors.*

Description

Conditional quantile estimation from k-nearest neighbors in the explanatory variable space.

Usage

```
kneigh.condquant(x, y, k = 10, p = 0.9)
```

Arguments

x	Matrix of explanatory (independent) variables of dimension d x n, d is the number of variables and n is the number of examples (patterns)
y	Vector of n dependent variables
k	Number of neighbors, default is 10.
p	Probability level, default is 0.99.

Details

For each example j (each column) in the matrix x, its k nearest neighbors in terms of Euclidean distance are identified. Let j1,..., jk be the k nearest neighbors. Then, the conditional quantile is estimated by computing the sample quantile over y[j1],...,y[jk].

Value

A vector of quantile of length n.

Author(s)

Julie Carreau

References

Bishop, C. (1995), Neural Networks for Pattern Recognition, Oxford

See Also

quantile

Examples

```
# generate train data
ntrain <- 500
xtrain <- runif(ntrain)
ytrain <- rfrechet(ntrain,loc = 3*xtrain+1,scale =
0.5*xtrain+0.001,shape=xtrain+1)
plot(xtrain,ytrain,pch=22) # plot train data
qgen <- qfrechet(0.99,loc = 3*xtrain+1,scale =
 0.5*xtrain+0.001,shape=xtrain+1) # compute quantile from generative model
points(xtrain,qgen,pch=".",col="orange")

kquant <- kneigh.condquant(t(xtrain),ytrain,p=0.99) # compute estimated quantile

points(xtrain,kquant,pch="o",col="blue")
# sample quantiles are not good in the presence of heavy-tailed data


ytrain <- rlnorm(ntrain,meanlog = 3*xtrain+1,sdlog =
0.5*xtrain+0.001)
dev.new()
plot(xtrain,ytrain,pch=22) # plot train data
qgen <- qlnorm(0.99,meanlog = 3*xtrain+1,sdlog =
 0.5*xtrain+0.001) # compute quantile from generative model
points(xtrain,qgen,pch=".",col="orange")
# compute estimated quantile
kquant <- kneigh.condquant(t(xtrain),ytrain,p=0.99)

points(xtrain,kquant,pch="o",col="blue") # a bit more reasonable
```

`lambertw` *Lambert W Function*

Description

The Lambert W function, given an input z, outputs w such that z = w exp(w). A numerical algorithm of order 4 is used to find the zero of z - w exp(w).

Usage

```
lambertw(z)
```

Arguments

z	the argument of the Lambert W function

Value

Returns the value of the Lambert W function.

Author(s)

Julie Carreau

References

Corless, R. M., Gonnet, G. H., Hare, D. E. G., Jeffrey, D. J. and Knuth, D. E. (1996), On the Lambert W Function, 5, Advances in Computational Mathematics

---

lognormixt *Mixture of Log-Normals*

Description

Density for a mixture of Log-Normals with `m` components.

Usage

```
dlognormixt(params, x, log = FALSE)
```

Arguments

`params`	matrix of mixture parameters of dimension 3 x m, where m is the number of components, so that each column contains the mixture parameters (pi, mu, sigma) related to one component
`x`	vector of sample data
`log`	logical, if TRUE, probabilities p are given as log(p).

Value

`dlognormixt` gives the density of the mixture of Log-Normals

Author(s)

Julie Carreau

See Also

`Lognormal`

`softplus` *Softplus Transform*

Description

The softplus (and inverse softplus) transform is useful to introduce positivity constraints on parameters of a function that will be optimized (e.g. MLE of the scale parameter of a density function). The softplus has been introduced to replace the exponential which might blow up for large argument. The softplus is given by : log(1+exp(x)) and converges to x for large values of x. Some care has been taken in the implementation of the softplus function to handle some numerical issues.

Usage

```
softplus(x)
softplusinv(y)
```

Arguments

`x`	is the value of the unconstrained parameter which is optimized
`y`	is the value of the positively constrained parameter

Details

Let sigma be the scale parameter of a density for which maximumm likelihood estimation will be performed. Then we can consider optimizing `softplusinv`(sigma) to ensure positivity of this parameter. Let sigma.unc be the optimzed unconstrained parameter, then `softplus`(sigma.unc) is the value of the MLE.

Value

The value of the softplus (or inverse softplus) transform.

Author(s)

Julie Carreau

References

Dugas, C., Bengio, Y., Belisle, F., Nadeau, C. and Garcia, R. (2001), A universal approximator of convex functions applied to option pricing, 13, Advances in Neural Information Processing Systems

Index