Package 'MoTBFs'

August 21, 2026

Title Learning Hybrid Bayesian Networks using Mixtures of Truncated Basis Functions

Version 2.0

Maintainer Ana D. Maldonado <ana.d.maldonado@ual.es>

Description Learning, manipulation and evaluation of mixtures of truncated basis functions (MoTBFs), which include mixtures of polynomials (MOPs) and mixtures of truncated exponentials (MTEs). MoTBFs are a flexible framework for modelling hybrid Bayesian networks (I. Pérez-Bernabé, A. Salmerón, H. Langseth (2015) <doi:10.1007/978-3-319-20807-7_36>; H. Langseth, T.D. Nielsen, I. Pérez-Bernabé, A. Salmerón (2014) <doi:10.1016/j.ijar.2013.09.012>; I. Pérez-Bernabé, A. Fernández, R. Rumí, A. Salmerón (2016) <doi:10.1007/s10618-015-0429-7>). The package provides functionality for learning univariate, multivariate and conditional densities, with the possibility of incorporating prior knowledge. Structural learning of hybrid Bayesian networks is also provided. A set of useful tools is provided, including plotting, printing and likelihood evaluation. This package makes use of S3 objects, with two new classes called 'motbf' and 'jointmotbf'.

License GPL-3

Imports quadprog, lpSolve, bnlearn, methods, ggm, Matrix, parallel, doParallel, foreach, infotheo

Suggests knitr, rmarkdown

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation no

Author Inmaculada Pérez-Bernabé [aut],
Antonio Salmerón [aut],
Thomas D. Nielsen [aut],
Ángel T. Sáez-Ruiz [aut],
Ana D. Maldonado [aut, cre] (ORCID:
<https://orcid.org/0000-0001-8253-2526>)

Repository CRAN

Date/Publication 2026-08-21 21:50:31 UTC

Contents

`asMOPString` *Parameters to MOP String*

Description

This function builds a string with the structure of a `'mop'` function.

Usage

```
asMOPString(parameters)
```

Arguments

parameters	A "numeric" vector containing the coefficients.

Value

A "character" string with a 'mop' structure.

Examples

```
param <- c(1,2,3,4)
asMOPString(param)

param <- 3.4
asMOPString(param)
```

---

asMTEString *Converting MTEs to strings*

Description

This function builds a string with the structure of an 'mte' function.

Usage

```
asMTEString(parameters, num = 5)
```

Arguments

parameters	A "numeric" vector containing the coefficients.
num	A "numeric" value which contains the denominator of the coefficient in the exponential.

Value

A "character" string with an 'mte' structure.

Examples

```
param <- -5.8
asMTEString(param)

param <- c(5.2,0.3,-3,4)
asMTEString(param)
```

BICMoTBF *Computing the BIC score of an MoTBF function*

Description

Computes the Bayesian information criterion value (BIC) of a mixture of truncated basis functions. The BIC score is the log likelihood penalized by the number of parameters of the function and the number of records of the evaluated data.

Usage

```
BICMoTBF(Px, X)
```

Arguments

Px	A function of class "motbf".
X	A "numeric" vector with the data to evaluate.

Value

A "numeric" value corresponding to the BIC score.

See Also

univMoTBF

Examples

```
## Data
X <- rexp(10000)

## Data test
Xtest <- rexp(1000)
Xtest <- Xtest[Xtest>=min(X) & Xtest<=max(X)]

## Learning
f1 <- univMoTBF(X, POTENTIAL_TYPE = "MOP", nparam = 10); f1
f2 <- univMoTBF(X, POTENTIAL_TYPE = "MTE", maxParam = 11); f2

## BIC values
BICMoTBF(Px = f1, X = Xtest)
BICMoTBF(Px = f2, X = Xtest)
```

`BICMultiFunctions` *BIC score for multiple functions*

Description

Compute the BIC score using more than one probability functions.

Usage

```
BICMultiFunctions(Px, X)
```

Arguments

`Px`	A list of objects of class `"motbf"`.
`X`	A list with as many `"numeric"` vectors as densities in `Px`, used to compute the BIC score for each density.

Value

The `"numeric"` BIC value.

See Also

univMoTBF

Examples

```
## Data
X <- rnorm(500)
Y <- rnorm(500, mean=1)
data <- data.frame(X=X, Y=Y)
## Data as a "list"
Xlist <- sapply(data, list)

## Learning as a "list"
Plist <- lapply(data, univMoTBF, POTENTIAL_TYPE="MOP")
Plist

## BIC value
BICMultiFunctions(Px=Plist, X=Xlist)
```

clean *Remove Objects from Memory*

Description

Clean the memory. Delete all the objects in memory and a garbage collection takes place.

Usage

```
clean(envir = globalenv(), n = 2)
```

Arguments

envir	The currently active environment; by default It is the gloval environment.
n	Number of garbage collection repetitions; by default n = 2.

Examples

```
## Run to clean the environment
clean()
clean(n=2)
```

coef.jointmotbf *Coefficients of a "jointmotbf" object*

Description

Extracts the parameters of a joint MoTBF density.

Usage

```
## S3 method for class 'jointmotbf'
coef(object, ...)
```

Arguments

object	An object of class "jointmotbf".
...	Other arguments, unnecessary for this function.

Value

A "numeric" vector with the parameters of the function.

See Also

jointmotbf.fit

Examples

```
## Generate a dataset
data <- data.frame(X1 = rnorm(100), X2 = rnorm(100))

## Joint function
dim <-c(2,4)
P <- jointmotbf.fit(data, dimensions = dim)
P$Time

## Coefficients
coef(P)
```

`coef.mop` *Extract coefficients from MOPs*

Description

It extracts the parameters of the learned mixtures of polynomial models.

Usage

```
coeffMOP(fx)

coeffPol(fx)
```

Arguments

`fx`	An `"motbf"` function of subclass `'mop'`.

Details

`coeffMOP()` return the coefficients of the terms in the function.

`coeffPol()` returns the coefficients of the potential of the polynomial basis in the function.

Value

An array with the parameters of the function.

See Also

coef.motbf and univMoTBF

Examples

```
## 1. EXAMPLE
data <- rchisq(1000, df=5)
fx1 <- univMoTBF(data, POTENTIAL_TYPE = "MOP")
hist(data, prob=TRUE, main="")
plot(fx1, xlim=range(data), col="red", add=TRUE)
coeffMOP(fx1) ## coef(fx1)
coeffPol(fx1)

## 2. EXAMPLE
data <- rexp(1000, rate=1/2)
fx2 <- univMoTBF(data, POTENTIAL_TYPE = "MOP")
hist(data, prob=TRUE, main="")
plot(fx2, xlim=range(data), col="red", add=TRUE)
coeffMOP(fx2) ## coef(fx2)
coeffPol(fx2)
```

`coef.motbf` *Extract the coefficients of an MoTBF*

Description

Extracts the parameters of the learned mixtures of truncated basis functions.

Usage

```
## S3 method for class 'motbf'
coef(object, ...)
```

Arguments

`object`	An object of class `motbf`.
`...`	other arguments.

Value

A numeric vector with the parameters of the function.

See Also

univMoTBF, coeffMOP and coeffMTE

Examples

```
## Data
X <- rchisq(2000, df = 5)

## Learning
f1 <- univMoTBF(X, POTENTIAL_TYPE = "MOP"); f1
## Coefficients
coef(f1)

## Learning
f2 <- univMoTBF(X, POTENTIAL_TYPE = "MTE", maxParam = 10); f2
## Coefficients
coef(f2)

## Learning
f3 <- univMoTBF(X, POTENTIAL_TYPE = "MOP", nparam=10); f3
## Coefficients
coef(f3)

## Plots
plot(NULL, xlim = range(X), ylim = c(0,0.2), xlab="X", ylab="density")
plot(f1, xlim = range(X), col = 1, add = TRUE)
plot(f2, xlim = range(X), col = 2, add = TRUE)
plot(f3, xlim = range(X), col = 3, add = TRUE)
hist(X, prob = TRUE, add= TRUE)
```

`coef.mte` *Extracting the coefficients of an MTE*

Description

It extracts the parameters of the learned mixtures of truncated exponential models.

Usage

```
coeffMTE(fx)

coeffExp(fx)
```

Arguments

`fx`	An `"motbf"` function of subclass `'mte'`.

Details

`coeffMOP()` return the coefficients of the terms in the function.

`coeffPol()` returns the coefficients of the potential of the exponential basis in the function.

Value

An array with the parameters of the function.

See Also

coef.motbf and univMoTBF

Examples

```
## 1. EXAMPLE
data <- rnorm(1000, mean=5)
fx1 <- univMoTBF(data, POTENTIAL_TYPE = "MTE")
hist(data, prob=TRUE, main="")
plot(fx1, xlim=range(data), col="red", add=TRUE)
coeffMTE(fx1) ## coef(fx1)
coeffExp(fx1)

## 2. EXAMPLE
data <- rexp(1000, rate=1/2)
fx2 <- univMoTBF(data, POTENTIAL_TYPE = "MTE")
hist(data, prob=TRUE, main="")
plot(fx2, xlim=range(data), col="red", add=TRUE)
coeffMTE(fx2) ## coef(fx2)
coeffExp(fx2)
```

---

coercion-motbf *Coerce MOTBF Objects to Character or Function*

Description

Converts 'motbf' and 'jointmotbf' objects into character string expressions or executable R functions.

Usage

```
## S3 method for class 'motbf'
as.character(x, ...)

## S3 method for class 'motbf'
as.function(x, ...)

## S3 method for class 'jointmotbf'
as.character(x, ...)

## S3 method for class 'jointmotbf'
as.function(x, ...)
```

Arguments

x	An object of class 'motbf' or 'jointmotbf'.
...	Further arguments passed to or from other methods. Not used currently.

Value

- as.character: Returns a character string representing the mathematical expression of the object.
- as.function: Returns an executable R function that accepts numeric arguments to evaluate the MoTBF expression.

Examples

```
## Example 1
X <- rchisq(5000, df = 3)
P <- univMoTBF(X, POTENTIAL_TYPE = "MOP"); P
as.function(P)(10)

## Example 2
data <- data.frame(X = rnorm(100), Y = rexp(100))
dim <- c(3,2)
P <- jointmotbf.fit(data, dimensions = dim)
density <- as.function(P)(data[,1], data[,2])
sum(log(density))
```

---

conditionalmotbf.learning

Learning conditional MoTBF densities

Description

Collection of functions used for learning conditional MoTBFs, computing the internal BIC, selecting the parents that get the best BIC value, and other internal functions required to learn the conditional densities.

Usage

```
conditionalMethod(
  data,
  nameParents,
  nameChild,
  numIntervals,
  POTENTIAL_TYPE,
  maxParam = NULL,
  s = NULL,
  priorData = NULL,
```

```
  scale = FALSE
)

conditional(
  data,
  nameParents,
  nameChild,
  domainChild,
  domainParents,
  numIntervals,
  mm,
  POTENTIAL_TYPE,
  maxParam = NULL,
  s = NULL,
  priorData = NULL,
  scale = FALSE
)

select(
  data,
  nameParents,
  nameChild,
  domainChild,
  domainParents,
  numIntervals,
  POTENTIAL_TYPE,
  maxParam = NULL,
  s = NULL,
  priorData = NULL,
  scale = FALSE
)

learn.tree.Intervals(
  data,
  nameParents,
  nameChild,
  domainParents,
  domainChild,
  numIntervals,
  POTENTIAL_TYPE,
  maxParam = NULL,
  s = NULL,
  priorData = NULL,
  scale = FALSE
)

BICscoreMoTBF(conditionalfunction, data, nameParents, nameChild)
```

Arguments

`data`	An object of class `"data.frame"`.
`nameParents`	A `"character"` vector containing the names of the parent variables.
`nameChild`	A `"character"` string containing the name of the child variable.
`numIntervals`	A positive integer indicating the maximum number of intervals for splitting the domain of the parent variables.
`POTENTIAL_TYPE`	A `"character"` string, either *MOP* or *MTE*, corresponding to the type of basis function.
`maxParam`	A positive integer which indicates the maximum number of coefficients in the function. If specified, the output is the function which gets the best BIC with, at most, this number of parameters. By default, it is set to `NULL`.
`s`	A `"numeric"` value indicating the expert's confidence in the prior knowledge. This argument takes values on the interval $[0, N]$, where N is the sample size, and is used to synchronize the support of the prior knowledge and the sample. By default, it is `NULL`, and must be modified only if prior information is to be incorporated in the learning process.
`priorData`	An object of class `"data.frame"`, corresponding to the prior information.
`scale`	A `"logical"` value indicating whether to standardize the numeric variables to have mean 0 and standard deviation 1.
`domainChild`	A `"numeric"` vector with the range of the child variable.
`domainParents`	An object of class `"matrix"` with the range of the parent variables, or a `"numeric"` vector if there is only one parent.
`mm`	One of the inputs and the output of the recursive internal function `"conditional"`.
`conditionalfunction`	The output of the internal function `learn.tree.Intervals`.

Details

The main function, `conditionalMethod()`, fits truncated basis functions for the conditioned variable for each configuration of splits of the parent variables. The domain of the parent variables is splitted in different intervals and univariate functions are fitted in these ranges. The remaining above described functions are internal to the main function.

Value

The main function `conditionalMethod` returns a list with the name of the parents, the different intervals and the fitted densities

See Also

printConditional

Examples

```
## Dataset
X <- rnorm(1000)
Y <- rbeta(1000, shape1 = abs(X)/2, shape2 = abs(X)/2)
Z <- rnorm(1000, mean = Y)
data <- data.frame(X = X, Y = Y, Z = Z)

## Conditional Method
parents <- c("X","Y")
child <- "Z"
intervals <- 2

potential <- "MTE"
fMTE <- conditionalMethod(data, nameParents = parents, nameChild = child,
numIntervals = intervals, POTENTIAL_TYPE = potential)
printConditional(fMTE)

#############################################################################

potential <- "MOP"
fMOP <- conditionalMethod(data, nameParents = parents, nameChild = child,
numIntervals = intervals, POTENTIAL_TYPE = potential, maxParam = 15)
printConditional(fMOP)

#############################################################################

#############################################################################
## Internal functions: Not needed to run ####################################
#############################################################################

domainP <- range(data[,parents])
domainC <- range(data[, child])
t <- conditional(data, nameParents = parents, nameChild = child,
domainParents = domainP, domainChild = domainC, numIntervals = intervals,
mm = NULL, POTENTIAL_TYPE = potential)
printConditional(t)
selection <- select(data, nameParents = parents, nameChild = child,
domainParents = domainP, domainChild = domainC, numIntervals = intervals,
POTENTIAL_TYPE = potential)
parent1 <- selection$parent; parent1
domainParent1 <- range(data[,parent1])
treeParent1 <- learn.tree.Intervals(data, nameParents = parent1,
nameChild = child, domainParents = domainParent1, domainChild = domainC,
numIntervals = intervals, POTENTIAL_TYPE = potential)
BICscoreMoTBF(treeParent1, data, nameParents = parent1, nameChild = child)


#############################################################################
#############################################################################
```

confusionMatrix *Confusion Matrix*

Description

Computes the confusion matrix for a given fitted model object.

Usage

```
confusionMatrix(x, ...)

## S3 method for class 'motbf_fit_cv'
confusionMatrix(x, digits = 4, ...)
```

Arguments

x	An object used to select the appropriate method, such as one of class 'motbf_fit_cv'.
...	Further arguments. Not used currently.
digits	An integer indicating the number of decimal places to round the results.

Value

A matrix or table containing the confusion matrix.

dataMining *Data pre-processing utilities*

Description

Collection of functions for discretizing, standardizing, converting factors to characters and other usufull methods for pre-processing datasets.

Usage

```
whichDiscrete(dataset, discreteVariables)

discreteVariables_as.character(dataset, discreteVariables)

standardizeDataset(dataset)

discretizeVariablesEWdis(dataset, numIntervals, factor = FALSE, binary = FALSE)

discreteVariablesStates(namevariables, discreteData)

nstates(DiscreteVariablesStates)
```

```
quantileIntervals(X, numIntervals)

scaleData(dataset, scale)
```

Arguments

dataset	A dataset of class "data.frame". Tha variables of the dataset can be discrete and continuous.
discreteVariables	A "character" array with the names of the discrete variables.
numIntervals	Number of bins used to discretize the continuous variables.
factor	A boolean value indicating if the variables should be considered as "factor" or as "character". By default it is set to FALSE.
binary	By default it is set to FALSE, indicating that only binary entries are used for continuous variables; a TRUE value means that binary entries are used to discretize the full dataset taking into account the states the discrete variables.
namevariables	an array with the names of the varibles.
discreteData	A discretized dataset of class "data.frame".
DiscreteVariablesStates	The output of the function discreteVariablesStates.
X	A "numeric" vector with the data values of a continuous variable.
scale	A "numeric" vector (when it refers to a single variable) or a "list" containing the name(s) of the variable(s) and the scale value.

Details

whichDiscrete() selects the position of the discrete variables.

discreteVariables_as.character() transforms the values of the discrete variables into character values.

standardizeDataset() standardizes all the variables in a data set.

discretizeVariablesEWdis() discretizes the continuous variables in a dataset using equal width binning.

discreteVariablesStates() extracts the states of the qualitative variables.

nstates() computes the number of different values of the discrete variables.

quantileIntervals() gets the quantiles of a variable taking into account the number of intervals into which its domain is splitted.

Examples

```
## dataset: 2 continuous variables, 1 discrete variable.
data <- data.frame(X = rnorm(100),Y = rexp(100,1/2), Z = as.factor(rep(c("s","a"), 50)))
disVar <- "Z" ## Discrete variable
class(data[,disVar]) ## factor
```

```
data <- discreteVariables_as.character(dataset = data, discreteVariables = disVar)
class(data[,disVar]) ## character

whichDiscrete(dataset = data, discreteVariables = "Z")

standData <- standardizeDataset(dataset = data)

disData <- discretizeVariablesEWdis(dataset = data, numIntervals = 3)

l <- discreteVariablesStates(namevariables = names(data), discreteData = disData)

nstates(DiscreteVariablesStates = l)

## Continuous variables
quantileIntervals(X = data[,1], numIntervals = 4)
quantileIntervals(X = data[,2], numIntervals = 10)
```

`derivMOP` *Derivative of a MOP*

Description

Compute the derivative of an `"motbf"` object with `'mop'` subclass.

Usage

```
derivMOP(fx)
```

Arguments

`fx`	An `"motbf"` object of the `'mop'` subclass.

Value

The derivative which is also an `"motbf"` function.

See Also

univMoTBF for learning and derivMoTBF for general `"motbf"` models.

Examples

```
## 1. EXAMPLE
X <- rexp(1000)
Px <- univMoTBF(X, POTENTIAL_TYPE="MOP")
derivMOP(Px)

## 2. EXAMPLE
X <- rnorm(1000)
```

```
Px <- univMoTBF(X, POTENTIAL_TYPE="MOP")
derivMOP(Px)

## Not run:
## 3. EXAMPLE
X <- rnorm(1000)
Px <- univMoTBF(X, POTENTIAL_TYPE="MTE")
derivMOP(Px)
## Error in derivMOP(Px): fx is an 'motbf' function but not 'mop' subclass.
class(Px)
subclass(Px)

## End(Not run)
```

`derivMoTBF` *Derivating MoTBFs*

Description

Compute the derivative of a one-dimensional mixture of truncated basis function.

Usage

```
derivMoTBF(fx)
```

Arguments

`fx`	An object of class `"motbf"`.

Value

The derivative of the MoTBF function, which is also an object of class `"motbf"`.

See Also

univMoTBF, derivMOP and derivMTE

Examples

```
## 1. EXAMPLE
X <- rexp(1000)
Px <- univMoTBF(X, POTENTIAL_TYPE="MOP")
derivMoTBF(Px)

## 2. EXAMPLE
X <- rnorm(1000)
Px <- univMoTBF(X, POTENTIAL_TYPE="MOP")
derivMoTBF(Px)

## 3. EXAMPLE
```

```
X <- rchisq(1000, df = 3)
Px <- univMoTBF(X, POTENTIAL_TYPE="MTE")
derivMoTBF(Px)

## Not run:
## 4. EXAMPLE
Px <- "x+2"
class(Px)
derivMoTBF(Px)
## Error in derivMoTBF(Px): "fx is not an 'motbf' function."

## End(Not run)
```

---

derivMTE *Derivating MTEs*

Description

Compute the derivative of an "motbf" object with 'mte' subclass.

Usage

```
derivMTE(fx)
```

Arguments

fx	An "motbf" object of the 'mte' subclass.

Value

The derivative which is also an "motbf" function.

See Also

univMoTBF for learning and derivMoTBF for general "motbf" models.

Examples

```
## 1. EXAMPLE
X <- rexp(1000)
Px <- univMoTBF(X, POTENTIAL_TYPE="MTE")
derivMTE(Px)

## 2. EXAMPLE
X <- rnorm(1000)
Px <- univMoTBF(X, POTENTIAL_TYPE="MTE")
derivMTE(Px)

## Not run:
## 3. EXAMPLE
```

```
X <- rnorm(1000)
Px <- univMoTBF(X, POTENTIAL_TYPE="MOP")
derivMTE(Px)
## Error in derivMTE(Px): fx is an 'motbf' function but not 'mte' subclass.
class(Px)
subclass(Px)

## End(Not run)
```

`dimensionFunction` *Dimension of MoTBFs*

Description

Get the dimension of `"motbf"` and `"jointmotbf"` densities. This function is deprecated; use `"getMotbfDim"` instead.

Usage

```
dimensionFunction(P)
```

Arguments

`P`	An object of class `"motbf"` and subclass 'mop' or `"jointmotbf"`.

Value

Dimension of the function.

See Also

univMoTBF and jointMoTBF

`discreteStatesFromBN` *Get the states of all discrete nodes from a MoTFB-BN*

Description

This function returns the states of all discrete node from a list obtained from motbf.fit.

Usage

```
discreteStatesFromBN(bn)
```

Arguments

`bn`	A list of lists obtained from motbf.fit.

Value

discreteStatesFromBN returns a list of length equal to the number of discrete nodes in the network. Each element of the list corresponds to a node and contains a character vector indicating the states of the node.

Examples

```
## Create a dataset
  # Continuous variables
  x <- rnorm(100)
  y <- rnorm(100)

  # Discrete variable
  z <- sample(letters[1:2],size = 100, replace = TRUE)

  data <- data.frame(C1 = x, C2 = y, D1 = z, stringsAsFactors = FALSE)

## Get DAG
  dag <- LearningHC(data)

## Learn a BN
  bn <- motbf.fit(dag, data, POTENTIAL_TYPE = "MTE")

## Get the states of the discrete nodes

  discreteStatesFromBN(bn)
```

---

ecoli *Data set Ecoli: Protein Localization Sites*

Description

This data set contains information of Escherichia coli. It is a bacterium of the genus Escherichia that is commonly found in the lower intestine of warm-blooded organism.

Format

A data frame with 336 rows, 8 variables and the class.

Details

Sequence Name Accession number for the SWISS-PROT database.

mcg McGeoch's method for signal sequence recognition.

gvh Von Heijne's method for signal sequence recognition.

lip Von Heijne's Signal Peptidase II consensus sequence score. Binary attribute.

chg Presence of charge on N-terminus of predicted lipoproteins. Binary attribute.

aac Score of discriminant analysis of the amino acid content of outer membrane and periplasmic proteins.

alm1 Score of the ALOM membrane spanning region prediction program.

alm2 Score of ALOM program after excluding putative cleavable signal regions from the sequence.

Class Class variable. 8 possibles states.

Source

http://archive.ics.uci.edu/ml/datasets/Ecoli

`eval.motbf` *Evaluation of MoTBFs*

Description

Evaluates a univariate (`"mop"`, `"mte"`) or joint distribution (`"jointmotbf"`) at a specific point.

Usage

```
eval.motbf(P, values)
```

Arguments

`P`	An object of class `"univmotbf"` or `"jointmotbf"`.
`values`	A list with the name of the variables equal to the values to be evaluated.

Value

If all the variables in the equation are evaluated, then a `"numeric"` value is returned. Otherwise, an `"univmotbf"` object or a `"jointmotbf"` object is returned.

Examples

```
#' ## 1. EXAMPLE
## Dataset with 2 variables
X <- data.frame(rnorm(100), rexp(100))

## Joint function
dim <- c(3,3) # dim <- c(5,4)
P <- jointmotbf.fit(X, dimensions = dim)

## Evaluation
val <- list(x = -1.5, y = 3)
eval.motbf(P, values = val)
val <- list(x = -1.5)
eval.motbf(P, values = val)
val <- list(y = 3)
eval.motbf(P, values = val)
```

```
##############################################################################
## MORE EXAMPLES #############################################################
##############################################################################

## Dataset with 3 variables
X <- data.frame(x = rnorm(100), y = rexp(100), z = rnorm(100, 1))

## Joint function
dim <- c(2,1,3)
P <- jointmotbf.fit(X, dimensions = dim)
P

## Evaluation
val <- list(x = 0.8, y = -2.1, z = 1.2)
eval.motbf(P, values = val)
val <- list(x = 0.8, y = 1.2)
eval.motbf(P, values = val)
val <- list(y = -2.1)
eval.motbf(P, values = val)
```

`evalJointFunction` *Evaluation of joint MoTBFs*

Description

Evaluates a `"jointmotbf"` object at a specific point. This function is deprecated; use `"eval.mop"` instead.

Usage

```
evalJointFunction(P, values)
```

Arguments

`P`	A `"jointmotbf"` object.
`values`	A list with the name of the variables equal to the values to be evaluated.

Value

If all the variables in the equation are evaluated then a `"numeric"` value is returned. Otherwise, an `"motbf"` object or a `"jointmotbf"` object is returned.

expectedValueMOP *Expected Value of an MoP Density Function*

Description

Computes the expected value (mean) of a Mixture of Polynomials (MoP) function over its domain.

Usage

```
expectedValueMOP(fx)
```

Arguments

fx	An object of class 'mop' representing an MoP probability density function.

Value

A numeric value representing the expected value of the distribution.

expectedValueMTE *Expected Value of an MTE Density Function*

Description

Computes the expected value (mean) of a Mixture of Truncated Exponentials (MTE) function over its domain.

Usage

```
expectedValueMTE(fx)
```

Arguments

fx	An object of class 'mte' representing an MTE probability density function.

Value

A numeric value representing the expected value of the distribution.

`findConditional` *Find Fitted Conditional MoTBFs*

Description

This function returns the conditional probability function of a node given an MoTBF-bayesian network and the value of its parents.

Usage

```
findConditional(node, bn, evi = NULL)
```

Arguments

node	A `character` string, representing the tardet variable.
bn	A list of lists obtained from MoTBFs_Learning, containing the conditional functions.
evi	A `data.frame` of dimension '1xn' that contains the values of the 'n' parents of the target node. This argument can be `NULL` if `"node"` is a root node.

Value

A list containing the conditional distribution of the target variable.

Examples

```
## Dataset
  data("ecoli", package = "MoTBFs")
  data <- ecoli[,-c(1,9)]

## Get directed acyclic graph
  dag <- LearningHC(data)

## Learn bayesian network
  bn <- MoTBFs_Learning(dag, data = data, numIntervals = 4, POTENTIAL_TYPE = "MTE")

## Specify the evidence set and node of interest
  evi <- data.frame(lip = "0.48", alm1 = 0.55, gvh = 1, stringsAsFactors=FALSE)
  node = "alm2"

## Get the conditional distribution
  findConditional(node, bn, evi)
```

generateNormalPriorData *Prior data generation*

Description

Generate a prior dataset taking in to account the relationships between the varibles in a given network.

Usage

```
generateNormalPriorData(graph, data, size, means, deviations = NULL)
```

Arguments

graph	A network of the class "bn", "graphNEL" or "network".
data	An object of class "data.frame" containing the continuous variables in the dataset.
size	A positive integer indicating the number of records to generate for each variable in the dataset.
means	A "numeric" vector with the average of the variables whose prior information is available. The names in the vector must be the same as the names of the variables in the data.frame.
deviations	A "numeric" vector with the standard deviations of the variables whose prior information is available. The names of the vector must be the same as the names of the variables in the data.frame. If not specified, the standard deviation of each variable is computed from 'data'.

Value

A normal prior data set of class "data.frame".

See Also

rnormMultiv

Examples

```
## Data
data(ecoli)
data <- ecoli[,-c(1,9)] ## remove sequece.name and class
X <- TrainingandTestData(data, percentage_test = 0.95)
Xtraining <- X$Training
Xtest <- X$Test

## DAG
dag <- LearningHC(data)
```

```
plot(dag)

## Means and desviations
colnames(data)

m <- sapply(data, function(x){ifelse(is.numeric(x), mean(x),NA)})
d <- sapply(data, function(x){ifelse(is.numeric(x), sd(x),NA)})


## Prior Dataset
n <- 5600
priorData <- generateNormalPriorData(dag, data = Xtraining, size = n, means = m)
summary(priorData)
ncol(priorData)
nrow(priorData)
class(priorData)
```

`getChildParentsFromGraph` *Get the list of relations in a graph*

Description

Compute the parents of each variable in the graph.

Usage

```
getChildParentsFromGraph(graph, nameVars = NULL)
```

Arguments

`graph`	A directed acyclic graph of the class `"graphNEL"`, `"network"` or `"bn"`.
`nameVars`	A character array containing the names of the variables in the graph. This parameter is only used when `graph` is of class `"network"`.

Value

A list where each element is a vector containing the name of a variable and its parents in the graph.

Examples

```
## Data
data(ecoli)
ecoli <- ecoli[,-1] ## Sequence Name

## DAG1
dag1 <- LearningHC(ecoli)
dag1
```

```
plot(dag1)
getChildParentsFromGraph(dag1)

## DAG2
dag2 <- LearningHC(ecoli, numIntervals = 10)
dag2
plot(dag2)
getChildParentsFromGraph(dag2)
```

`getCoefficients` *Get the coefficients*

Description

Compute the coefficients for the linear opinion pool

Usage

```
getCoefficients(fPI, rangeNewPriorData, fD, data, domain, coeffversion)
```

Arguments

`fPI`	The function fitted to the prior data, of class `"motbf"`.
`rangeNewPriorData`	An array of length 2 with the new domain of the prior function.
`fD`	The function fitted to the original data, of class `"motbf"`.
`data`	A `"numeric"` array which contains the sample.
`domain`	A `"numeric"` array with the domain of the data density function.
`coeffversion`	A `"numeric"` value between `1--4` which contains the used version for computing the coefficients in the linear opinion pool to combine the prior function and the data function. By default `coeffversion = "4"` is used, so the combination depends on the goodness of the model versus another random positive MoTBF model.

Details

`coeffversion` can be: `"1"` coef1 and coef2 are the sum of the probabilities of one of the function over the sum of all probabilities, respectively; `"2"` coef1 and coef2 are the solution of a linear optimization problem which tries to maximize the sum 1 for each row of probabilities; `"3"` coef1 and coef2 are the difference of the log-likelihood of the evaluated model and a random uniform model over the sum of both differences, respectively; `"4"` coef1 and coef2 are the difference of the log-likelihood of the evaluated model and a ramdom positive MoTBF model over the sum of both differences, respectively.

Value

A "numeric" value of length 2 giving the coefficients which are the weigth of the two function to combine.

See Also

learnMoTBFpriorInformation

Examples

```
## Data
X <- rnorm(15)

## Prior Data
priordata <- rnorm(5000)

## Learning
confident <- 5
type <- "MOP"
f <- learnMoTBFpriorInformation(priorData = priordata, data = X, s = confident,
POTENTIAL_TYPE = type, returnAll = TRUE)
attributes(f)

## Coefficients: linear opinion pool
getCoefficients(fPI = f$priorFunction, rangeNewPriorData = f$domain, fD = f$dataFunction,
data = X, domain = range(X), coeffversion = 4)

getCoefficients(fPI = f$priorFunction, rangeNewPriorData = f$domain, fD = f$dataFunction,
data = X, domain = range(X), coeffversion = 1)

getCoefficients(fPI = f$priorFunction, rangeNewPriorData = f$domain, fD = f$dataFunction,
data = X, domain = range(X), coeffversion = 3)

getCoefficients(fPI = f$priorFunction, rangeNewPriorData = f$domain, fD = f$dataFunction,
data = X, domain = range(X), coeffversion = 2)
```

---

getDAG *Retrieve DAG from BN*

Description

Get the Directed acyclic graph (DAG) from a Bayesian network (BN)

Usage

```
getDAG(bn)
```

Arguments

bn	An object of class `motbf_fit` or `bn.fit` (from bnlearn package).

Value

An object of class bn (same class as in bnlearn package).

---

getMotbfDim *Extract Dimension of MoTBFs*

Description

Get the dimension of "motbf" and "jointmotbf" densities.

Usage

```
getMotbfDim(P)
```

Arguments

P	An object of class "motbf" and subclass 'mop' or "jointmotbf".

Value

Dimension of the function.

See Also

univMoTBF and jointmotbf.fit

Examples

```
## 1. EXAMPLE
## Data
X <- rnorm(2000)

## Univariate function
f <- univMoTBF(X, POTENTIAL_TYPE = "MOP")
getMotbfDim(f)

## 2. EXAMPLE
## Dataset with 2 variables
X <- data.frame(x = rnorm(100), y = rnorm(100))

## Joint function
dim <- c(2,3)
P <- jointmotbf.fit(X, dimensions = dim)

## Dimension of the joint function
```

```
getMotbfDim(P)
```

---

getMotbfVar *Extract Variables of MoTBFs*

Description

Get the names of the variables involved in a "univmotbf" or jointmotbf object.

Usage

```
getMotbfVar(P)
```

Arguments

P	An object of class "univmotbf" or "jointmotbf".

Value

A "character" vector with the names of the variables in the function.

Examples

```
# 1. EXAMPLE
## Generate a dataset
data <- data.frame(X1 = rnorm(100), X2 = rnorm(100))

## Joint function
dim <-c(3,2)
P <- jointmotbf.fit(data, dimensions = dim)
P

## Variables
getMotbfVar(P)

#############################################################################
## MORE EXAMPLES ############################################################
#############################################################################

## Generate a dataset
data <- data.frame(X1 = rnorm(100), X2 = rnorm(100), X3 = rnorm(100))

## Joint function
dim <- c(2,1,3)
P <- jointmotbf.fit(data, dimensions = dim)

## Variables
getMotbfVar(P)
```

getNonNormalisedRandomMoTBF *Ramdom MoTBF*

Description

Generates a non normalized (i.e. not integrating to 1) positive MoTBF function.

Usage

```
getNonNormalisedRandomMoTBF(degree, POTENTIAL_TYPE = "MOP")
```

Arguments

degree	A "numeric" value containing the degree of the random function.
POTENTIAL_TYPE	A "character" string specifying the posibles potential types, must be one of "MOP" or "MTE".

Value

A "numeric" vector of length 2 giving the coefficients.

Examples

```
getNonNormalisedRandomMoTBF(8, POTENTIAL_TYPE = "MOP")
getNonNormalisedRandomMoTBF(11, POTENTIAL_TYPE = "MTE")
```

getStructure *Hybrid Bayesian Network structure learning*

Description

Learn the structure of a hybrid Bayesian network, using a fixed method (Naive Bayes, NB), a restricted method (Tree augmented Naive Bayes, TAN), or an unrestricted method (the **hill climbing**, HC, score-based local search method).

Usage

```
getStructure(data, method, target = NULL)
```

Arguments

data	A dataset with discrete and continuous variables. If the discrete variables are not of class "factor", they are automatically converted.
method	A "character" string indicating the method to learn the structure: NB (naive Bayes), TAN (Tree augmented Naive Bayes), or HC (hill climbing) are the available options.
target	An optional parameter only used in the case of NB and TAN to specify the class variable.

Details

getStructure() automatically converts non-numeric variables into factors before calling function hc() from the bnlearn package. In the case of TAN, it converts all numeric and non-numeric variables into factors (using 4 equal width intervals) before calling tree.bayes() from the bnlearn package.

Value

The output is a "bn" object containing the learned graph.

See Also

hc

Examples

```
## Data
data(ecoli)
ecoli <- ecoli[,-1] ## Sequence Name

## DAG1
dag1 <- getStructure(ecoli, method = "HC")
dag1
plot(dag1)

## DAG2
dag2 <- getStructure(ecoli, method = "TAN", target = "mcg")
dag2
plot(dag2)
```

`get_approx_posterior` *Approximate inference*

Description

`get_approx_posterior()` returns an approximation to the posterior probability distribution of a target variable given a set of observed variables. The inference process is based on sample generation. See details.

Usage

```
get_approx_posterior(
  bn,
  target,
  evidence = NULL,
  size = 100,
  parallel = FALSE,
  ...
)
```

Arguments

`bn`	An object of class `motbf_fit`, obtained from function motbf.fit.
`target`	A character string equal to the name of the variable of interest.
`evidence`	A `data.frame` of one row containing the value of the observed variables.
`size`	A non-negative integer giving the number of random samples to generate from `bn`.
`parallel`	`logical` indicating if the particle generation should be parallelized. As a default, it is set to FALSE.
`...`	Optional arguments passed on to the `univMoTBF` function. `evalRange`, `nparam` and `maxParam` can be specified. `POTENTIAL_TYPE` is taken from the 'bn' object.

Details

If any node is observed, i.e., argument `evidence` is not NULL, samples are generated from the Bayesian network using the likelihood weighting algorithm. Otherwise, i.e., no node is observed, samples are generated using the forward sampling algorithm.

Value

A list of two elements: 1) the posterior probability distribution of the target variable, and 2) a data.frame with the generated sample, whose weights are attached as an attribute called weights (if `evidence` is not NULL).

References

Henrion, M. (1988). Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In Machine Intelligence and Pattern Recognition (Vol. 5, pp. 149-163). North-Holland.

Examples

```
## Dataset
  data("ecoli", package = "MoTBFs")
  data <- ecoli[,-c(1,9)]

## Get directed acyclic graph
  dag <- LearningHC(data)

## Learn bayesian network
  bn <- motbf.fit(dag, data = data, numIntervals = 4, POTENTIAL_TYPE = "MOP")

## Specify the evidence set and target variable
  obs <- data.frame(lip = "0.48", alm1 = 0.55, gvh = 1, stringsAsFactors=FALSE)
  node <- "alm2"

## Get the posterior distribution of 'node' given "evidence" and the generated sample
  get_approx_posterior(bn, target = node, evidence = obs, size = 10, maxParam = 15)
```

---

goodnessMoTBFBN *BIC of a hybrid BN*

Description

Compute the BIC score and the loglikelihood from the fitted MoTBFs functions in a hybrid Bayesian network, i.e., from objects of class motbf.fit.

Usage

```
logLikelihood.MoTBFBN(object, data)

BiC.MoTBFBN(object, data)
```

Arguments

`object`	The output of the 'motbf.fit()' function
`data`	The dataset of class `data.frame`.

Value

A numeric value giving the log-likelihood of the BN.

See Also

MoTBFs_Learning

Examples

```
## Dataset Ecoli
require(MoTBFs)
data(ecoli)
data <- ecoli[,-c(1)] ## remove variable sequence

## Directed acyclic graph
dag <- LearningHC(data)

## Learning BN
intervals <- 3
potential <- "MOP"
P1 <- MoTBFs_Learning(graph = dag, data = data, POTENTIAL_TYPE=potential,
numIntervals = intervals, maxParam = 5)
logLikelihood.MoTBFBN(P1, data) ##BIC$LogLikelihood
BIC <- BiC.MoTBFBN(P1, data)
BIC$BIC

## Learning BN
intervals <- 2
potential <- "MTE"
P2 <- MoTBFs_Learning(graph = dag, data = data, POTENTIAL_TYPE=potential,
numIntervals = intervals, maxParam = 10)
logLikelihood.MoTBFBN(P2, data) ##BIC$LogLikelihood
BIC <- BiC.MoTBFBN(P2, data)
BIC$BIC
```

---

integralJointMoTBF *Integration with MoTBFs*

Description

Integrate a "jointmotbf" object over an non defined domain. It is able to get the integral of a joint function over a set of variables or over all the variables in the function. This function is deprecated; use "integrate.motbf" instead.

Usage

```
integralJointMoTBF(P, var = NULL)
```

Arguments

P	A "jointmotbf" object.
var	A "character" vector containing the name of the variables that will be integrated out. Instead of the names, the position of the variables can be given. By default it's NULL then all the variables are integrated out.

Value

A multiintegral of a joint function of class "jointmotbf".

---

integralMOP *Integration of MOPs*

Description

Method to calculate the non-defined integral of an "motbf" object of 'mop' subclass. This function is deprecated; use "integrate.motbf" instead.

Usage

```
integralMOP(fx)
```

Arguments

fx	An "motbf" object of subclass 'mop'.

Value

The non-defined integral of the function.

See Also

univMoTBF for learning and integralMoTBF for a more complete function to get defined and non-defined integrals of class "motbf".

---

integralMoTBF *Integrating MoTBFs*

Description

Compute the integral of a one-dimensional mixture of truncated basis function over a bounded or unbounded interval. This function is deprecated; use "integrate.motbf" instead.

Usage

```
integralMoTBF(fx, min = NULL, max = NULL)
```

Arguments

fx	An object of class "motbf".
min	The lower integration limit. By default it is NULL.
max	The upper integration limit. By default it is NULL.

Details

If the limits of the interval, min and max are NULL, then the output is the expression of the indefinite integral. If only 'min' contains a numeric value, then the expression of the integral is evaluated at this point.

Value

`integralMoTBF()` returns either the indefinite integral of the MoTBF function, which is also an object of class `"motbf"`, or the definite integral, wich is a `"numeric"` value.

See Also

univMoTBF, integralMOP and integralMTE

`integralMTE` *Integrating MTEs*

Description

Method to calculate the non-defined integral of an `"motbf"` object of `'mte'` subclass. This function is deprecated; use `"integrate.motbf"` instead.

Usage

```
integralMTE(fx)
```

Arguments

`fx`	An `"motbf"` object of subclass `'mte'`.

Value

The non-defined integral of the function.

See Also

univMoTBF for learning and integralMoTBF for a more complete function to get defined and non-defined integrals of class `"motbf"`.

`integrate.motbf` *Integrating MoTBFs*

Description

Compute the integral of a one-dimensional mixture of truncated basis function (objects of class `"mop"` or `"mte"`) over a bounded or unbounded interval, or compute the indefinite integral of a joint function (object of class `"jointmotbf"`) over a subset of variables or over all the variables in the function.

Usage

```
integrate.motbf(f, ...)
```

Arguments

`f`	An object of class `"mop"`, `"mte"` or `"jointmotbf"`.
`...`	optional arguments to be passed to subsequent methods for the integrate.motfb() function. See details.

Details

This function is a wrapper of internal functions integrateMOP(), integrateMTE() and integrateJointmotbf().

If f is of class `"mop"` or `"mte"`, valid optional arguments are 'lower' (the lower integration limit) and 'upper' (the upper integration limit), which represent the limits of the interval to compute the definite integral. If 'lower' and 'upper' are not specified, then the output is the expression of the indefinite integral.

On the other hand, if f is of class `"jointmotbf"`, the only valid optional argument is 'var', which is a `"character"` vector containing the name of the variables that will be integrated out. If not specified, then all the variables are integrated out.

Note that integrate.motfb() deprecates the following functions, included in previous versions of the package: integralMTE, integralMOP, integralMoTBF and integralJointMoTBF.

Value

If f is of class `"mop"` or `"mte"`, `integrate.motfb()` returns either the indefinite integral of the MoTBF function, which is also an object of classes `"motbf"`, `"univmotbf"`, and either `"mop"` or `"mte"`); or the definite integral, which is a `"numeric"` value. If f is of class `"jointmotbf"`, `integrate.motfb()` returns a multi-integral of the joint function, which is also of class `"jointmotbf"`.

See Also

univMoTBF and jointMoTBF

Examples

```
## 1. EXAMPLE
## Univariate MOP integral
X <- rexp(1000)
fx <- univMoTBF(X, POTENTIAL_TYPE = "MOP", scale = FALSE)
integrate.motbf(fx)
integrate.motbf(fx, 2, 6)


## 2. EXAMPLE
## Univariate MOP integral and plot of result
Y <- rnorm(1000)
fy <- univMoTBF(Y, POTENTIAL_TYPE = "MOP")
Fy <- integrate.motbf(fy)
plot(Fy)
integrate.motbf(fy, min(Y), max(Y))

## 3. EXAMPLE
## Univariate MTE integral
Z <- rchisq(1000, df = 3)
fz <- univMoTBF(Z, POTENTIAL_TYPE = "MTE", scale = FALSE)
integrate.motbf(fz)
integrate.motbf(fz, lower = 2, upper = 5)

## Not run:
## 4. EXAMPLE
Px <- "1+x+5"
class(Px)
integrate.motbf(Px)
## Error in integrate.motbf(Px) : Argument "f" is not of class "motbf"

## End(Not run)

## 5. EXAMPLE: Joint MOP integral
## Dataset with 2 variables
data <- data.frame(x = rnorm(100), y = rnorm(100))

## Joint function
dim <- c(2, 3)
P <- jointmotbf.fit(data, dimensions = dim)

## Integral
integrate.motbf(P)
integrate.motbf(P, var = "x")
integrate.motbf(P, var = "y")

#############################################################################
## MORE EXAMPLES ############################################################
#############################################################################

## Dataset with 3 variables
data <- data.frame(x = rnorm(50), y = rnorm(50), z = rnorm(50))
```

```
## Joint function
dim <- c(2,2,3)
P = jointmotbf.fit(data, dimensions = dim)

## Integral
integrate.motbf(P)
integrate.motbf(P, var="x")
integrate.motbf(P, var=c("x","z"))
```

`is.discrete` *Check discreteness of a node*

Description

This function allows to check whether a node is discrete or not

Usage

```
is.discrete(node, bn)
```

Arguments

node	A character (name of node) or numeric (index of node in the bn list) input.
bn	A list of lists obtained from MoTBFs_Learning.

Value

`is.discrete` returns TRUE or FALSE depending on whether the node is discrete or not.

Examples

```
## Create a dataset
  # Continuous variables
  x <- rnorm(100)
  y <- rnorm(100)

  # Discrete variable
  z <- sample(letters[1:2],size = 100, replace = TRUE)

  data <- data.frame(C1 = x, C2 = y, D1 = z, stringsAsFactors = FALSE)

## Get DAG
  dag <- LearningHC(data)

## Learn BN
```

```
  bn <- MoTBFs_Learning(dag, data, POTENTIAL_TYPE = "MTE")

## Check wheter a node is discrete or not

  # Using its name
  is.discrete("D1", bn)

  # Using its index position
  is.discrete(3, bn)
```

`is.motbf` *Check MoTBF Classes and Subclasses*

Description

Utility functions to check whether an object belongs to a specific MoTBF class, or to identify its underlying subclass (`'mop'` or `'mte'`).

Usage

```
is.motbf(x, class = "motbf")

is.univmotbf(x, class = "univmotbf")

is.jointmotbf(x, class = "jointmotbf")

is.motbf_fit(x, class = "motbf_fit")

is.motbf_fit_cv(x, class = "motbf_fit_cv")

is.mte(x)

is.mop(x)

subclass(fx)
```

Arguments

`x`	An object to be checked.
`class`	Character string specifying the target class name.
`fx`	An object to determine the subclass for.

Value

- `is.*`: Logical value (`TRUE` or `FALSE`) indicating if the object belongs to the checked class.
- `subclass`: A character string (`"mop"` or `"mte"`) specifying the underlying family of the object.

is.observed *Observed Node*

Description

`is.observed()` checks whether a node belongs to the evidence set or not.

Usage

```
is.observed(node, evi)
```

Arguments

node	A `character` string, matching the node's name.
evi	A `data.frame` of the evidence set.

Value

This function returns TRUE if "node" is included in "evi", or, otherwise, FALSE.

Examples

```
## Data frame of the evidence set
  obs <- data.frame(lip = "1", alm2 = 0.5, stringsAsFactors=FALSE)

## Check if x is in obs
  is.observed("x", obs)
```

is.root *Root nodes*

Description

`is.root` checks whether a node has parents or not.

Usage

```
is.root(node, dag)
```

Arguments

node	A character string indicating the node's name.
dag	An object of class `"bn"`.

Value

`is.root` returns TRUE or FALSE depending on whether the node is root or not.

Examples

```
## Create a dataset
  # Continuous variables
  x <- rnorm(100)
  y <- rnorm(100)

  # Discrete variable
  z <- sample(letters[1:2],size = 100, replace = TRUE)

  data <- data.frame(C1 = x, C2 = y, D1 = z, stringsAsFactors = FALSE)

## Get DAG
  dag <- LearningHC(data)

## Check if a node is root
 is.root("C1", dag)
```

`jointmotbf.fit` *Joint MoTBF density learning*

Description

Function for learning joint MoTBFs. The `jointmotbf.fit()` function is a wrapper of two internal (non-exported) functions: `getParamJoint()` and `fixParamJoint()`. The first one gets the parameters by solving a quadratic optimization problem, minimizing the mean squared error between the empirical joint CDF and the estimated CDF. The density is obtained as the derivative of the estimated CDF. The second one, `fixParamJoint()`, fixes the equation of the joint function using the previously learned parameters and converting this `"character"` string into an object of class `"jointmotbf"`.

Usage

```
jointmotbf.fit(
  X,
  ranges = NULL,
  dimensions = NULL,
  fitPoints = 10,
  constraints = 10
)
```

Arguments

`X`	a dataset of class `"data.frame"`.
`ranges`	a `"numeric"` matrix containing the range of the variables used to fit the function, where each column corresponds to a variable. If not specified, the range of each variable is computed from the data.
`dimensions`	a `"numeric"` vector containing the number of parameters of each variable.

`fitPoints`	an `"integer"` indicating the number of points per variable to use to build the expanded grid where the objective function will be evaluated when optimizing the parameters.
`constraints`	an `"integer"` indicating the number of constraints under which to minimize the quadratic function.

Value

`jointmotbf.fit()` returns a list with the following elements:

`Function`	The analytical expression of the learned density.
`Domain`	A `"matrix"` containing the domain of each variable over which the density is defined.
`Iterations`	The number of iterations needed to solve the problem.
`Time`	The execution time.

Examples

```
## 1. EXAMPLE
## Generate a multinormal dataset
data <- data.frame(X1 = rnorm(100), X2 = rnorm(100))

## Joint learnings
dim <- c(2,3)
P <- jointmotbf.fit(data, dimensions = dim)

P
attributes(P)
class(P)

##############################################################################
## MORE EXAMPLES #############################################################
##############################################################################

## Generate a dataset
data <- data.frame(X1 = rnorm(100), X2 = rnorm(100), X3 = rnorm(100))

## Joint learnings
dim <- c(3,2,3)
P <- jointmotbf.fit(data, dimensions = dim)
P
attributes(P)
class(P)
```

`jointmotbf.learning` *Joint MoTBF density learning*

Description

Deprecated functions; use `"jointmotbf.fit"` instead.

Usage

```
parametersJointMoTBF(X, ranges = NULL, dimensions = NULL)

jointMoTBF(object)
```

Arguments

`X`	A dataset of class `"data.frame"`.
`ranges`	A `"numeric"` matrix containing the range of the varibles used to fit the function, where each column corresponds to a variable. If not specified, the range of each variable is computed from the data.
`dimensions`	A `"numeric"` vector containing the number of parameters of each varible.
`object`	A list with the output of the function `parametersJointMoTBF()`.

Details

Two functions for learning joint MoTBFs. The first one, `parametersJointMoTBF()`, gets the parameters by solving a quadratic optimization problem, minimizing the mean squared error between the empirical joint CDF and the estimated CDF. The density is obtained as the derivative od the estimated CDF. The second one, `jointMoTBF()`, fixes the equation of the joint function using the previously learned parameters and converting this `"character"` string into an object of class `"jointmotbf"`.

Value

`parametersJointMoTBF()` returns a list with the following elements: **Parameters**, which contains the computed coefficients of the resulting function; **Dimension**, which is a `"numeric"` vector containing the number of coefficients used for each variable; **Range** contains a `"numeric"` matrix with the domain of each variable, by columns; **Iterations** contains the number of iterations needed to solve the problem; **Time** contains the execution time.

`jointMoTBF()` returns an object of class `"jointmotbf"`, which is a list whose only visible element is the analytical expression of the learned density. It also contains the other aforementioned elements, which can be retrieved using `attributes()`

`LearningHC` *Score-based hybrid Bayesian Network structure learning*

Description

Learn the structure of a hybrid Bayesian network using the **hill climbing** local search method.

Usage

```
LearningHC(dataset, numIntervals = NULL)
```

Arguments

`dataset`	A dataset with discrete and continuous variables. If the discrete variables are not of class `"factor"`, they are automatically converted.
`numIntervals`	A `"numeric"` value indicating the number of categories used when discretizing a continuous variable, corresponding to intervals of equal width. By default it is `NULL`, meaning that the continuous variables are not discretized.

Details

`LearningHC()` automatically converts non-numeric variables into factors before calling function `hc()` from the `bnlearn` package. `LearningHC()` can also be used to discretize the dataset, using the equal width method, before calling `hc()`.

Value

The output is a `"bn"` object containing the learned graph.

See Also

hc

Examples

```
## Data
data(ecoli)
ecoli <- ecoli[,-1] ## Sequence Name

## DAG1
dag1 <- LearningHC(ecoli)
dag1
plot(dag1)

## DAG2
dag2 <- LearningHC(ecoli, numIntervals = 10)
dag2
plot(dag2)
```

`learnMoTBFpriorInformation` *Incorporating prior knowledge in the estimation process*

Description

Learns a univariate MoTBF function using prior information.

Usage

```
learnMoTBFpriorInformation(
  priorData,
  data,
  s,
  POTENTIAL_TYPE,
  domain = range(data),
  coeffversion = 4,
  restrictDomain = TRUE,
  maxParam = NULL,
  returnAll = FALSE,
  scale = TRUE
)
```

Arguments

`priorData`	A `"numeric"` vector which contains the prior information.
`data`	A `"numeric"` vector containing the observed data.
`s`	A `"numeric"` value which specifies the expert confidence in the prior knowledge. This argument takes values on the interval $[0, N]$, where N is the sample size, and is used to synchronize the support of the prior knowledge and the sample.
`POTENTIAL_TYPE`	A `"character"` string, either *MOP* or *MTE*, corresponding to the type of basis function.
`domain`	A `"numeric"` vector which contains the bounding values to fit the function. By default, it is the range of the data.
`coeffversion`	A `"numeric"` value between `1--4` which contains the used version for computing the coefficients of the linear opinion pool to combine the prior function and the data function. By default, `coeffversion = "4"` is used, so the combination depends on the goodness of the model versus another random model.
`restrictDomain`	A logical value. This argument allows to choose if the domain is used joining both domains, the prior one and the data domain or trimming them. By default, `TRUE` is used, so the domain will be trimmed.
`maxParam`	A positive integer which indicates the maximum number of coefficients in the function. If specified, the output is the function which gets the best BIC with, at most, this number of parameters. By default, it is set to `NULL`.

`returnAll` A logical value indicating whether to return all prior, data and posterior functions (TRUE) or only the posterior (FALSE).

`scale` A "logical" value indicating whether to standardize the numeric variables to have mean 0 and standard deviation 1.

Value

If returnAll = TRUE, the function returns a list with the elements

`coeffs` An "numeric" vector with the two coefficients of the linear opinion pool

`posteriorFunction` The final function after combining.

`priorFunction` The fit of the prior data.

`dataFunction` The fit of the original data.

`rangeNewPriorData` A "numeric" vector which contains the final domain where the functions are defined.

See Also

getCoefficients

Examples

```
## Data
X <- rnorm(15)

## Prior Data
priordata <- rnorm(5000)

## Test data
test <- rnorm(1000)
testData <- test[test>=min(X)&test<=max(X)]

## Learning
type <- "MOP"
confident <- 3 ## confident <- 1,2,...,length(X)
f <- learnMoTBFpriorInformation(priorData = priordata, data = X, s = confident,
POTENTIAL_TYPE = type, returnAll = TRUE)
attributes(f)

## Log-likelihood
sum(log(as.function(f$dataFunction)(testData)))
sum(log(as.function(f$posteriorFunction)(testData))) ## best loglikelihood
```

`marginal.jointmotbf` *Marginalization of MoTBFs*

Description

Computes the marginal densities from a `"jointmotbf"` object.

Usage

```
marginal.jointmotbf(P, var)
```

Arguments

`P`	An object of class `"jointmotbf"`, i.e., the joint density function.
`var`	A vector containing the names of the marginal variables (those being retained in 'P'). This argument accepts the `"numeric"` position (w.r.t. P$Domain) or the `"character"` name of the variables.

Value

If 'var' is an atomic vector, the marginal distribution of the variable specified in 'var' is computed from the joint distribution and the result is an object of class `"univmotbf"`, i.e., a univariate distribution.

If 'var' contains more than one variable, the joint distribution over this subset is computed, i.e., the variables NOT contained in 'var' are marginalized out and the result is an object of class `"jointmotbf"`.

See Also

jointmotbf.fit and eval.motbf

Examples

```
## 1. EXAMPLE
## Dataset with 2 variables
data <- data.frame(x = rnorm(100), y = rnorm(100))

## Joint function
dim <- c(4,3)
P <- jointmotbf.fit(data, dimensions = dim)

## Marginal
marginal.jointmotbf(P, var = "x")
marginal.jointmotbf(P, var = 2)

##############################################################################
## MORE EXAMPLES #############################################################
##############################################################################
```

```
## Generate a dataset with 3 variables
data <- data.frame(x = rnorm(100), y = rnorm(100), z = rnorm(100))

## Joint function
dim <- c(2,2,3)
P <- jointmotbf.fit(data, dimensions = dim)

## Marginal
marginal.jointmotbf(P, var = "x")
marginal.jointmotbf(P, var = "y")
marginal.jointmotbf(P, var = c("x", "z"))
```

`marginalJointMoTBF` *Marginalization of MoTBFs*

Description

Computes the marginal densities from a `"jointmotbf"` object. This function is deprecated; use `"marginal.jointmotbf"` instead.

Usage

```
marginalJointMoTBF(P, var)
```

Arguments

`P`	An object of class `"jointmotbf"`, i.e., the joint density function.
`var`	The `"numeric"` position or the `"character"` name of the marginal variable.

Value

The marginal of a `"jointmotbf"` function. The result is an object of class `"motbf"`.

See Also

jointMoTBF and evalJointFunction

mop.learning *Fitting mixtures of polynomials*

Description

These functions fit mixtures of polynomials (MOPs). Least square optimization is used to minimize the quadratic error between the empirical cumulative distribution and the estimated one.

Usage

```
mop.learning(X, nparam, domain)

bestMOP(X, domain, maxParam = NULL)
```

Arguments

X	A "numeric" data vector.
nparam	Number of parameters of the function.
domain	A "numeric" containing the interval over which the function is defined.
maxParam	A "numeric" value indicating the maximum number of coefficients in the function. By default it is NULL, which means that the number of parameter is not limited. The output is the function which gets the best BIC (with at mostmaxParam parameters if not NULL).

Details

mop.learning(): The returned value $Function is the only visible element which contains the algebraic expression. Using attributes the name of the others elements are shown and also they can be extracted with $. The summary of the function also shows all these elements.

bestMOP(): The first returned value $bestPx contains the output of the mop.learning() function with the number of parameters which gets the best BIC values, taking into account the BIC score to penalize the functions. It evaluates the two next functions, if the BIC score does not improve then the function with the last best BIC is returned.

Value

mop.lerning() returns a list of n elements:

Function	An "motbf" object of the 'mop' subclass.
Subclass	'mop'.
Domain	The range where the function is defined to be a legal density function.
Iterations	The number of iterations that the optimization problem takes to minimize the errors.
Time	The CPU time employed.

bestMOP() returns a list including the polynomial function with the best BIC score, the number of parameters and an array with the BIC values of the evaluated functions.

See Also

univMoTBF A complete function for learning MOPs which includes extra options.

Examples

```
## 1. EXAMPLE
data <- rnorm(1000)

## MOP with fix number of degrees
fx <- mop.learning(data, nparam=7, domain=range(data))
fx
hist(data, prob=TRUE, main="")
plot(fx, col=2, xlim=range(data), add=TRUE)

## Best MOP in terms of BIC
fMOP <- bestMOP(data, domain=range(data))
attributes(fMOP)
fMOP$bestPx
hist(data, prob=TRUE, main="")
plot(fMOP$bestPx, col=2, xlim=range(data), add=TRUE)

## 2. EXAMPLE
data <- rbeta(4000, shape1=1/2, shape2=1/2)

## MOP with fix number of degrees
fx <- mop.learning(data, nparam=6, domain=range(data))
fx
hist(data, prob=TRUE, main="")
plot(fx, col=2, xlim=range(data), add=TRUE)

## Best MOP in terms of BIC
fMOP <- bestMOP(data, domain=range(data), maxParam=6)
attributes(fMOP)
fMOP$bestPx
attributes(fMOP$bestPx)
hist(data, prob=TRUE, main="")
plot(fMOP$bestPx, col=2, xlim=range(data), add=TRUE)
```

MOPTAN *Fitting MoTBFs TAN models*

Description

Perform a TAN model of class MoTBF based on maximizing the Mutual Information.

Usage

```
fit_tan(
  target,
```

```
  data,
  fit.args = NULL,
  root = NULL,
  all = FALSE,
  mutualInfoCond = NULL,
  parallel = FALSE
)

mutual_information_tan(data, target, fit.args = NULL, parallel = FALSE)
```

Arguments

target	A character string indicating the name of the target or class variable. Target must be a column of data and it can be a continuous or discrete variable.
data	A data.frame containing the variables, which can contain continuous and discrete variables.
fit.args	A list a list containing optional arguments used to fit the models. These arguments must be those accepted by function motbf.fit, i.e., 'numIntervals' (4), 'POTENTIAL_TYPE' ('MOP'), 'maxParam' (7), 's' (NULL), 'priorData' (NULL) or 'scale' (TRUE). If fit.args is left NULL, the default values (in brackets) for those arguments will be used.
root	A character string indicating the label of the root predictor variable of TAN model.
all	A logical flag. If TRUE, the function return a list which contains two elements: the TAN model and the mutual information used to compute the maximum spanning tree. Defaults to FALSE.
mutualInfoCond	A numeric matrix indicating the estimation of the mutual information coefficientes for Chow-Liu- algorithm in TAN. If it is NULL, it is computed using mutual_information_tan function.
parallel	A logical flag. If TRUE, computation runs in parallel using foreach and doParallel. Defaults to FALSE.

Details

The main function, fit_tan(), fits a MoTBF Tree Augmented Naive Bayes model using the specified data.

Value

The main function, fit_tan(), returns an object of class "motbf_fit". When all=TRUE, it returns a list with the Bayesian network and the mutual information matrix used to compute the maximun spanning tree.

Function mutual_information_tan() returns a symmetric numeric matrix of dimensions $k \times k$, where k is the number of predictor variables. Row and column names correspond to the predictor variables, and the entries contain the estimated conditional mutual information values. This matrix is used to compute the TAN model.

Examples

```
data = iris
data$Species = as.factor(data$Species)
# Fit TAN model for classification
tan = fit_tan("Species",data)
```

`MoTBF-Distribution` *Random generation for MoTBF distributions*

Description

Random generation for mixtures of truncated basis functions defined in a specific domain. The inverse transform method is used.

Usage

```
rMoTBF(size, fx, domain = NULL)

inversionMethod(size, fx, domain = NULL, data = NULL)
```

Arguments

`size`	A non-negative integer indicating the number of records to generate.
`fx`	An object of class `"motbf"`.
`domain`	A `"numeric"` vector indicating the lower and upper limits to sample from. If not specified, the range is taken from the object `fx`.
`data`	A `"numeric"` vector to be compared with the simulated sample. By default, it is `NULL`; otherwise, the empirical cumulative distributions of both the data and the simulated sample are plotted and the Kolmogorov Smirnov test is used to test whether or not both samples can be considered to be drawn from the same distribution.

Value

`rMoTBF()` returns a `"numeric"` vector containing the simulated values. `inversionMethod()` returns a list with the simulated values and the results of the two-sample Kolmogorov-Smirnov test, as well as the plot of the CDFs of the original and simulated data.

Examples

```
## 1. EXAMPLE
## Data
X <- rnorm(1000, mean = 5, sd = 3)

## Learning
```

```
f <- univMoTBF(X, POTENTIAL_TYPE="MOP", nparam=10)
plot(f, xlim = f$Domain)

## Random sample
Y <- rMoTBF(size = 500, fx = f)
ks.test(X,Y)

## Plots
hist(Y, prob = TRUE, add = TRUE)


## 2. EXAMPLE
## Data
X <- rweibull(5000, shape=2)

## Learning
f <- univMoTBF(X, POTENTIAL_TYPE="MOP", nparam=10)
plot(f, xlim = f$Domain)

## Random sample
inv <- inversionMethod(size = 500, fx = f, data = X)
attributes(inv)
inv$test
Y <- inv$sample

## Plots
plot(f, xlim = f$Domain)
hist(Y, prob = TRUE, add = TRUE)
```

`motbf.cv` *Cross-validation for MoTBFs*

Description

Perform a k-fold cross validation for Bayesian networks of class MoTBF.

Usage

```
motbf.cv(
  data,
  dag,
  loss,
  k = 10,
  target = NULL,
  seed = NULL,
  fit.args = NULL,
  loss.args = NULL,
```

```
  foldsIndex = NULL,
  ...
)
```

Arguments

data	an object of class "data.frame", which can contain continuous and discrete variables.
dag	a network of the class "bn", "graphNEL" or "network".
loss	a character string indicating which loss function should be used. Currently, two options are available: 'logl', for the log-likelihood of the model; and 'pred', for the predictive error. See details.
k	an integer indicating the number of folds to split the data set. If k = 0, the train and test sets are the same data; if k = 1, hold-out validation is carried out, i.e., the data set is split in train (80% by default) and test (20% by default); finally, if k >=2, k-fold cross validation is carried out.
target	a character string indicating which node is the target. This argument might be NULL if the loss function chosen is the log-likelihood of the model ('logl').
seed	an integer to specify the seed. The k-folds are created randomly, so one might expect slightly different results unless 'seed' is used.
fit.args	a list containing optional arguments used to fit the models. These arguments must be those accepted by function motbf.fit, i.e., 'numIntervals' (4), 'POTENTIAL_TYPE' ('MOP'), 'maxParam' (NULL), 's' (NULL), 'priorData' (NULL) or 'scale' (TRUE). If fit.args is left NULL, the default values (in brackets) for those arguments will be used.
loss.args	a list containing optional arguments related to the loss functions. Currently available arguments are: 'loss.matrix' and 'percentage_test'. See details.
foldsIndex	a list containing the test indexes for each fold of cross-validation. If is not null, k and seed are ignored.
...	Additional arguments. Not used currently.

Details

If the basis function is MTE, the loss function is the log-likelihood of the model. On the other hand, if the basis function if MOP, the loss function might be either the predictive error ('pred') or the the log-likelihood of the model ('logl').

Details on the loss argument:

'logl' The log-likelihood of the model is computed. This measure is available for both basis functions, MTE and MOP.

'pred' This option is only available for MOPs. The predictive error (root mean squared error) or classification accuracy is computed as the loss function, depending on the nature of the target variable (continuous or discrete, respectively). The program will guess which type the target variable is and compute the corresponding measure.

Details on the loss.args argument. Currently, two arguments can be specified within this list:

'p.test' Only used if k = 1. This argument specifies the proportion of the data set that goes to the test set (between 0 and 1).

'loss.matrix' A squared matrix used to compute a weighted classification accuracy for discrete targets. This matrix is multiplied by the confusion matrix element by element, and the resulting matrix is used to compute the classification accuracy. The loss matrix allows to increase the penalty of some user-specified errors. If the loss matrix provided is a constant matrix of ones, the result is the standard classification accuracy. Note that the diagonal of the loss matrix is regarded as a reward, while the off-diagonal is regarded as a cost.

Value

An object of class "motbf.fit.cv". This is a list of "k" elements, each of them containing the results of each fold. More specifically, each element contains:

test a "data.frame" of the subset used to fit the model.

fitted an object of class "motbf.fit", i.e., the model fitted.

loss the loss value computed for the fold. If "pred" is chosen for the argument "loss", then each element of the "motbf.fit.cv" object also contains:

predicted a vector of predictions for the target variable.

observed a vector of observed records of the target variable.

Examples

```
#################
### EXAMPLE 1 ###
#################
## Perform 2-fold cross validation using the default model arguments
## and the log-likelihood as loss function

# Load data
data(ecoli)
ecoli <- ecoli[,-c(1,9)]

# Learn DAG
dag <- LearningHC(ecoli)

# Run cross validation
cv = motbf.cv(data = ecoli, dag, k = 2, loss = 'logl')
cv


#################
### EXAMPLE 2 ###
#################
## Choose different arguments to fit the model parameters
fit.args = list(numIntervals = 3, POTENTIAL_TYPE = 'MOP', maxParam = 4)

# Run cross validation using the classification accuracy as loss function
cv = motbf.cv(data = ecoli, dag, k = 2, loss = 'pred', target = 'lip',
  fit.args = fit.args)
```

```
cv
summary(cv)

#################
### EXAMPLE 3 ###
#################

## Specify a loss matrix to increase the penalty of classification errors
lossFunctionMatrix = matrix(c(c(1,2), c(3, 1)),nrow = 2, ncol = 2, byrow = TRUE)

# Run cross validation using the weighted classification accuracy as loss function
cv = motbf.cv(data = ecoli, dag, k = 2, loss = 'pred', target = 'lip',
  loss.args = list(loss.matrix = lossFunctionMatrix))
cv
summary(cv)

#################
### EXAMPLE 4 ###
#################

# Run cross validation using the root mean squared error as loss function
cv = motbf.cv(data = ecoli, dag, k = 2, loss = 'pred', target = 'mcg')
cv
```

`motbf.fit` *Learning hybrid BNs with MoTBFs*

Description

Learn mixtures of truncated basis functions in a full hybrid network.

Usage

```
motbf.fit(
  graph,
  data,
  numIntervals = 4,
  POTENTIAL_TYPE = "MOP",
  maxParam = NULL,
  s = NULL,
  priorData = NULL,
  scale = TRUE
)

MoTBFs_Learning(
  graph,
  data,
  numIntervals = 4,
```

```
  POTENTIAL_TYPE = "MOP",
  maxParam = NULL,
  s = NULL,
  priorData = NULL,
  scale = TRUE
)
```

Arguments

graph	A network of the class "bn" (from bnlearn package).
data	An object of class "data.frame"; it can contain continuous and discrete variables.
numIntervals	A positive integer indicating the maximum number of intervals for splitting the domain of the continuous parent variables. By default, it is set to 4.
POTENTIAL_TYPE	A "character" string, either *MOP* or *MTE*, corresponding to the type of basis function. By default, it is set to *MOP*.
maxParam	A positive integer which indicates the maximum number of coefficients in the function. If specified, the output is the function which gets the best BIC with, at most, this number of parameters. By default, it is set to NULL.
s	A "numeric" value which specifies the expert confidence in the prior knowledge. This argument takes values on the interval $[0, N]$, where N is the sample size, and is used to synchronize the support of the prior knowledge and the sample. By default, it is NULL, and must be modified only if prior information is to be incorporated to the fits.
priorData	An object of class "data.frame", corresponding to the prior information.
scale	A "logical" value indicating whether to standardize the numeric variables to have mean 0 and standard deviation 1.

Details

If the variable is discrete then it computes the probabilities and the size of each leaf. Children that have discrete parents have as many functions as configurations of the parents. Children that have continuous parents have as many functions as the number indicated in the argument "numIntervals" for each parent. Children that have mixed parents, combine both methods. The BIC criterion is used to decide the number of splitting points of the parent domains and to choose the number of basis functions used.

Value

A list of lists. Each list contains two elements

Child	A "charater" string which contains the name of the child variable.
functions	A list of three elements: the name of the parents; a "numeric" vector indicating the interval of the parent; and the fitted function in this interval.

Examples

```
## Dataset Ecoli
require(MoTBFs)
data(ecoli)
data <- ecoli[,-c(1)] ## remove variable sequence

## Directed acyclic graph
dag <- LearningHC(data)

## Learning BN
intervals <- 3
potential <- "MOP"
bn1 <- motbf.fit(graph = dag, data = data, numIntervals = intervals,
POTENTIAL_TYPE = potential, maxParam = 5)
bn1

## Learning BN
intervals <- 4
potential <- "MTE"
bn2 <- motbf.fit(graph = dag, data = data, numIntervals = intervals,
POTENTIAL_TYPE = potential, maxParam = 15)
bn2
```

`motbf2bnlearn` *Export discrete motbf to bnlearn format*

Description

Export a discrete BN created with motbf.fit to an object of class "bn" of package bnlearn.

Usage

```
motbf2bnlearn(bn)
```

Arguments

bn	An object of class `motbf_fit`, obtained from function motbf.fit. Only discrete networks are valid.

Value

An object of class bn.

motbf2grain *Export discrete motbf to grain format*

Description

Export a discrete BN created with motbf.fit to an object of class "grain" of package gRain

Usage

```
motbf2grain(bn)
```

Arguments

bn	An object of class motbf_fit, obtained from function motbf.fit. Only discrete networks are valid.

Value

An object of class grain.

motbf_type *Type of MoTBF*

Description

This function checks whether the density functions of a MoTBF-BN are of type MTE or MOP.

Usage

```
motbf_type(bn)
```

Arguments

bn	A list of lists obtained from the function MoTBFs_Learning.

Value

A character string, specifying the subclass of MoTBF, i.e., either MTE or MOP.

Examples

```
## Dataset
  data("ecoli", package = "MoTBFs")
  data <- ecoli[,-c(1,9)]

## Get directed acyclic graph
  dag <- LearningHC(data)

## Learn bayesian network
  bn <- MoTBFs_Learning(dag, data = data, numIntervals = 4, POTENTIAL_TYPE = "MTE")

## Get MoTBF sub-class
  motbf_type(bn)
```

`mte.learning` *Fitting mixtures of truncated exponentials.*

Description

These functions fit mixtures of truncated exponentials (MTEs). Least square optimization is used to minimize the quadratic error between the empirical cumulative distribution function and the estimated one.

Usage

```
mte.learning(X, nparam, domain)

bestMTE(X, domain, maxParam = NULL)
```

Arguments

`X`	A `"numeric"` data vector.
`nparam`	Number of parameters of the resulting density function.
`domain`	A `"numeric"` containing the domain if the function to estimate.
`maxParam`	A `"numeric"` value indicating the maximum number of coefficients in the function. By default it is `NULL`; otherwise, the output is the function which gets the best BIC with at most this number of parameters.

Details

`mte.learning()`: The returned value `$Function` is the only visible element which contains the algebraic expression. Using attributes the name of the others elements are shown and also they can be abstract with `$`. The summary of the function also shows all this elements.

`bestMTE()`: The first returned value `$bestPx` contains the output of the `mte.learning()` function with the number of parameters which gets the best BIC value, taking into account the Bayesian information criterion (BIC) to penalize the functions. It evaluates the two next functions, if the BIC doesn't improve then the function with the last best BIC is returned.

Value

`mte.lerning()` returns a list of n elements:

`Function`	An `"motbf"` object of the `'mte'` subclass.
`Subclass`	`'mte'`.
`Domain`	The range where the function is defined to be a legal density function.
`Iterations`	The number of iterations that the optimization problem employed to minimize the errors.
`Time`	The CPU time consumed.

`bestMTE()` returns a list including the MTE function with the best BIC score, the number of parameters, the best BIC value and an array contained the BIC values of the evaluated functions.

See Also

univMoTBF A complete function for learning MoTBFs which includes extra options.

Examples

```
## 1. EXAMPLE
data <- rchisq(1000, df=3)

## MTE with fix number of parameters
fx <- mte.learning(data, nparam=7, domain=range(data))
hist(data, prob=TRUE, main="")
plot(fx, col=2, xlim=range(data), add=TRUE)

## Best MTE in terms of BIC
fMTE <- bestMTE(data, domain=range(data))
attributes(fMTE)
fMTE$bestPx
hist(data, prob=TRUE, main="")
plot(fMTE$bestPx, col=2, xlim=range(data), add=TRUE)

## 2. EXAMPLE
data <- rexp(1000, rate=1/3)

 ## MTE with fix number of parameters
fx <- mte.learning(data, nparam=8, domain=range(data))
## Message: The nearest function with odd number of coefficients
hist(data, prob=TRUE, main="")
plot(fx, col=2, xlim=range(data), add=TRUE)

## Best MTE in terms of BIC
fMTE <- bestMTE(data, domain=range(data), maxParam=10)
attributes(fMTE)
fMTE$bestPx
attributes(fMTE$bestPx)
hist(data, prob=TRUE, main="")
plot(fMTE$bestPx, col=2, xlim=range(data), add=TRUE)
```

newRangePriorData *Redefining the Domain*

Description

Computes the new domain of two datasets.

Usage

```
newRangePriorData(fPI, priorData, N, domain, s, POTENTIAL_TYPE)
```

Arguments

fPI	The function fitted to the prior data, of class "motbf".
priorData	A "numeric" array with the values to be included as prior information.
N	A "numeric" value equal to the data size.
domain	A "numeric" array with the domain of the data density.
s	A "numeric" value which is the expert's confidence on the prior information. It is a number between 0 and the data size.
POTENTIAL_TYPE	A "character" string giving the potential of the model, i.e. "MOP" if the basis functions are polynomials, or "MTE" if they are exponentials.

Value

A "numeric" array which contains the new domain of the prior function.

Examples

```
## Data
X <- rnorm(15)

## Prior Data
priordata <- rnorm(5000)

## Learning
type = "MTE"
fPrior <- univMoTBF(priordata, POTENTIAL_TYPE = type)

## New range
confident <- 5 ## confident <- 1,2,...,length(X)
domain <- range(X)
N <- length(X)
newRange <- newRangePriorData(fPrior, priorData = priordata, N = N,
domain = domain, s = confident, POTENTIAL_TYPE = type)
newRange
```

nVariables *Number of Variables in a Joint Function*

Description

Compute the number of variables which are in a `jointmotbf` object. This function is deprecated; use `"getMotbfVar"` instead.

Usage

```
nVariables(P)
```

Arguments

`P`	An `"motbf"` object or a `"jointmotbf"` object.

Value

A `"character"` vector with the names of the variables in the function.

plot.motbf *Plots for 'motbf' objects*

Description

Draws an `'motbf'` function.

Usage

```
## S3 method for class 'motbf'
plot(x, ...)

## S3 method for class 'univmotbf'
plot(x, xlim = NULL, ylim = NULL, type = "l", add = FALSE, ...)

## S3 method for class 'piecewisemop'
plot(x, xlim = NULL, ylim = NULL, ...)

## S3 method for class 'motbf.fit.node'
plot(x, panels = TRUE, ...)

## S3 method for class 'jointmotbf'
plot(
  x,
  type = "contour",
  ranges = NULL,
```

```
  orientation = c(5, -30),
  data = NULL,
  filled = TRUE,
  ticktype = "simple",
  main = NULL,
  ...
)
```

Arguments

x	An object of class 'motbf' or one of its subclasses ('univmotbf', 'piecewisemop', 'motbf.fit.node', or 'jointmotbf').
...	Further arguments to be passed as for plot.
xlim, ylim	Numeric vectors of length 2 specifying x and y axis limits. By default 0:1. Used in 'univmotbf' and 'piecewisemop'.
type	Character string specifying the plot type, as for plot. For 'univmotbf' defaults to 'l' (line); for 'jointmotbf' defaults to "contour" (another option is "perspective").
add	Logical; if TRUE, adds the plot to the existing active graphics device. Used only in 'univmotbf'.
panels	Logical; if TRUE, displays plots in separate panels. Used only in 'motbf.fit.node'.
ranges	A "numeric" matrix containing the domain of the variables, by columns, which is used to specify the plotting range. Used only in 'jointmotbf'.
orientation	A "numeric" vector indicating the perpective of the plot in degrees. By default, it is set to (5,-30). Used only in 'jointmotbf'.
data	An object of class "data.frame" containing two columns only. This argument is used to draw the points over the main plot. By default, it is set to NULL. Used only in 'jointmotbf'.
filled	A logical argument; it is only used if type = "contour". is active. By default, it is TRUE, so filled contours are plotted. Used only in 'jointmotbf'.
ticktype	A "character" string, either *simple* or *detailed*. By default, it is set to "simple", which draws just an arrow parallel to the axis to indicate direction of increase. In contrast, "detailed" draws normal ticks. This argument is only used in the "perspective" plot. Used only in 'jointmotbf'.
main	Title string for the plot. Used in 'jointmotbf'.

Value

A plot of the specificated function.

Examples

```
## 1. univmotbf - Example for univariate distributions
## Data
X <- rexp(2000)
```

```
f1 <- univMoTBF(X, POTENTIAL_TYPE = "MOP")
f2 <- univMoTBF(X, POTENTIAL_TYPE = "MTE", maxParam = 10)

## Plots
plot(NULL, xlim = range(X), ylim = c(0,0.8), xlab="X", ylab="density")
plot(f1, xlim = range(X), col = 1, add = TRUE)
plot(f2, xlim = range(X), col = 2, add = TRUE)


## 2. jointmotbf - Example for join distributions
## Data
X <- data.frame(rnorm(500), rnorm(500))

## Joint function
dim <- c(3,3)
P <- jointmotbf.fit(X, dimensions = dim)

## Plots
plot(P)
plot(P, type = "perspective", orientation = c(90,0))
```

`plotConditional` *Plot Conditional Functions*

Description

Plot conditional MoTBF densities.

Usage

```
plotConditional(
  conditionalFunction,
  data,
  nameChild = NULL,
  points = FALSE,
  color = NULL,
  ...
)
```

Arguments

`conditionalFunction`	the output of function conditionalMethod. A list containing the the interval of the parent and the final conditional density (MTE or MOP).
`data`	An object of class `data.frame`, corresponding to the dataset used to fit the conditional density.
`nameChild`	A `character` string, corresponding to the name of the child variable in the conditional density. By default, it is `NULL`.

points	A logical value. If TRUE, the sample points are overlaid.
color	If not specified, a default palette is used.
...	Additional graphical parameters passed to filled.contour().

Details

If the number of parents is greater than one, then the error message "It is not possible to plot the conditional function." is reported.

Value

A plot of the conditional density function.

See Also

conditionalMethod

Examples

```
## Data
X <- rnorm(1000)
Y <- rnorm(1000, mean=X)
data <- data.frame(X=X,Y=Y)
cov(data)

## Conditional Learning
parent <- "X"
child <- "Y"
intervals <- 5
potential <- "MTE"
P <- conditionalMethod(data, nameParents=parent, nameChild=child,
numIntervals=intervals, POTENTIAL_TYPE=potential)
plotConditional(conditionalFunction=P, data=data)
plotConditional(conditionalFunction=P, data=data, points=TRUE)
```

---

predict.motbf_fit *Predict from an MoTBF Bayesian Network*

Description

Predict from an MoTBF Bayesian Network

Usage

```
## S3 method for class 'motbf_fit'
predict(
  object,
  target,
  data,
  method = NULL,
  prob = FALSE,
  parallel = FALSE,
  ...
)
```

Arguments

object	An object of class "motbf_fit"
target	A character string indicating the variable to be predicted
data	A data frame containing the predictive variables
method	A character string indicating the method to perform the inference. Options are NULL, "ve" (variable elimination) and "fs" (forward sampling).
prob	A logical value. If TRUE, the probabilities used for classification are returned as an attribute.
parallel	A logical value. If TRUE, parallelization is carried out.
...	Additional arguments used if method = "fs".

---

preprocessedData *Data cleaning*

Description

Delete rows of a dataset which contains anomalous values.

Usage

```
preprocessedData(data, strangeElements)
```

Arguments

data	A dataset of class "matrix" or "data.frame",
strangeElements	A "character" string which contains the elements to remove.

print.motbf *Print object of class motbf* print *method for class* "motbf".

Description

Print object of class motbf print method for class "motbf".

Print a single node of a BN. This function is called by print.motbf_fit, but not exported.

Print the results of a k-fold cross validation

Usage

```
## S3 method for class 'motbf'
print(x, ...)

## S3 method for class 'motbf_fit'
print(x, ...)

## S3 method for class 'motbf.fit.node'
print(x, ...)

## S3 method for class 'motbf_fit_cv'
print(x, ...)

## S3 method for class 'univmotbf'
print(x, ...)

## S3 method for class 'piecewisemop'
print(x, ...)

## S3 method for class 'jointmotbf'
print(x, ...)
```

Arguments

x	An object of class "motbf", "motbf_fit", "univmotbf", "piecewisemop", "jointmotbf", or "motbf.fit.cv".
...	optional arguments passed to print for other classes created in the MoTBFs package. Currently, no optional arguments are supported.

Details

The following classes are created in the MoTBFs package:

motbf the generic class common to all objects created in the package

univmotbf the class corresponding to MTE or MOP univariate distributions. An object of class univmotbf is the output of function univMoTBF().

piecewisemop the class corresponding to MOP univariate distributions defined by multiple subfunctions with different domain. When calling `variableElimination()`, the output might be of this class.

jointmotbf the class corresponding to joint distributions. An object of class `jointmotbf` is the output of function `jointMOP()`

motbf_fit the class corresponding to fully fitted Bayesian network models (either discrete, continuous or hybrid). An object of class `motbf_fit` is the output of function `motbf.fit()`.

motbf.fit.cv the class corresponding to k-fold cross validation results. An object of class `motbf.fit.cv` is the output of function `motbf.cv()`.

motbf.fit.node the class corresponding to a single node of a Bayesian Network.

Examples

```
## Dataset Ecoli
data(ecoli)
data <- ecoli[,-c(1)] ## remove variable sequence

## Directed acyclic graph
dag <- LearningHC(data)

## Learning BN
P <- motbf.fit(graph = dag, data = data, numIntervals = 3, POTENTIAL_TYPE = "MOP",
maxParam = 15)
P
```

`printConditional` *Summary of conditional MoTBF densities*

Description

Print the description of an MoTBF demnsity for one variable conditional on another variable.

Usage

```
printConditional(conditionalFunction)
```

Arguments

`conditionalFunction`
: the output of the function `conditionalMethod`. A list with the interval of the parent and the final `"motbf"` density function fitted in each interval.

Value

The results of the conditional function are shown.

See Also

conditionalMethod

Examples

```
## Data
X <- rexp(500)
Y <- rnorm(500, mean=X)
data <- data.frame(X=X,Y=Y)
cov(data)

## Conditional Learning
parent <- "X"
child <- "Y"
intervals <- 5
potential <- "MOP"
P <- conditionalMethod(data, nameParents=parent, nameChild=child,
numIntervals=intervals, POTENTIAL_TYPE=potential)
printConditional(P)
```

`probDiscreteVariable` *Probability distribution of discrete variables*

Description

Compute the probabilities of a discrete variable from a data set using Laplace correction.

Usage

```
probDiscreteVariable(x)
```

Arguments

`x`	a `"factor"` containing the records of the discrete variable.

Details

Laplace correction is used to avoid the zero probability problem, with $\alpha = 1$ as smoothing factor. Therefore, the probability for each state, θ_i, is computed as

$$\theta_i = \frac{x_i + 1}{N + K}$$

for $i = 0, 1, \ldots, K$, where x_i is the number of records of state i, N is the total number of records, and K is the total number of states.

Value

A list of 2 elements:

coeff	a named vector that contains the probabilities.
sizeDataLeaf	a vector containing the number of records in each leaf of the discrete tree.

See Also

discreteVariablesStates

Examples

```
## Simulate discrete variable
x <- factor(sample(c('yes', 'no', 'maybe'), 500, replace = TRUE),
  levels = c('yes', 'no', 'maybe'))

## Compute probabilities
p <- probDiscreteVariable(x)
p
```

---

query *Conditional probability queries*

Description

Compute conditional probability queries from a sample.

Usage

```
query(sample, event, evidence)
```

Arguments

sample	A data.frame containing a sample.
event	A expression describing the event of interest
evidence	A expression describing the conditioning evidence.

Value

The conditional probability: P(event|evidence).

Examples

```
data("ecoli", package = "MoTBFs")
dat <- ecoli[,-c(1,4,5,9)]

# Build DAG
dag <- LearningHC(dat)

# Learn BN parameters
bn = motbf.fit(dag, dat)

# Get sample from bn
sam = sample_motbfs(bn, 50)

# Compute P(mcg > 0.6 | aac < 0.8 & alm2 < 0.3)
query(sam, event = (mcg >0.6), evidence = (aac<0.8 & alm2 <0.3))
```

`r.data.frame` *Initialize Data Frame*

Description

The function `r.data.frame()` initializes a data frame with as many columns as nodes in the MoTBF-network. It also asings each column its data type, i.e., numeric or character. In the case of character columns, the states of the variable are extracted from the `"bn"` argument and included as levels.

Usage

```
r.data.frame(bn)
```

Arguments

bn	A list of lists obtained from the function MoTBFs_Learning.

Value

An object of class `"data.frame"`, which contains the data type of each column and has no rows.

Examples

```
## Create a dataset
  # Continuous variables
  x <- rnorm(100)
  y <- rnorm(100)

  # Discrete variable
  z <- sample(letters[1:2],size = 100, replace = TRUE)
```

```
  data <- data.frame(C1 = x, C2 = y, D1 = z, stringsAsFactors = FALSE)

## Get DAG
  dag <- LearningHC(data)

## Learn a BN
  bn <- motbf.fit(dag, data, POTENTIAL_TYPE = "MTE")

## Initialize a data.frame containing 3 columns (x, y and z) with their attributes.
  r.data.frame(bn)
```

`rescaledFunctions` *Rescaling MoTBF functions*

Description

A collection of function to reescale an MoTBF function to the original offset and scale. This is useful when data was standardized previously to learning.

Usage

```
rescaledMoTBFs(fx, data = NULL)

rescaledMOP(fx, data = NULL)

rescaledMTE(fx, data)

ToStringRe_MTE(parameters, data, num = 1)

meanMOP(fx)

rescale_motbf_fit(object, POTENTIAL_TYPE, data)
```

Arguments

`fx`	A function of class `"motbf"` learned from a scaled data.
`data`	the original data set (non-scaled)
`parameters`	A `"numeric"` vector with the coefficients to create the rescaled MoTBF.
`num`	A `"numeric"` value which contains the denominator of the coefficient in the exponential. By default it is 5.
`object`	and object of class motbf_fit that has been learned with scaled data
`POTENTIAL_TYPE`	the potential of the model (either "MOP" or "MTE")

Value

An `"motbf"` function of the original data.

See Also

univMoTBF

Examples

```
## 1. EXAMPLE
X <- rchisq(1000, df = 8) ## data
modX <- scale(X) ## scale data

## Learning
f <- univMoTBF(modX, POTENTIAL_TYPE = "MOP", nparam=10)
plot(f, xlim = range(modX), col=2)
hist(modX, prob = TRUE, add = TRUE)

## Rescale
origF <- rescaledMoTBFs(f, X)
plot(origF, xlim = range(X), col=2)
hist(X, prob = TRUE, add = TRUE)
expectedValueMOP(origF)
mean(X)

## 2. EXAMPLE
X <- rweibull(1000, shape = 20, scale= 10) ## data
modX <- as.numeric(scale(X)) ## scale data

## Learning
f <- univMoTBF(modX, POTENTIAL_TYPE = "MTE", nparam = 9)
plot(f, xlim = range(modX), col=2, main="")
hist(modX, prob = TRUE, add = TRUE)

## Rescale
origF <- rescaledMoTBFs(f, X)
plot(origF, xlim = range(X), col=2)
hist(X, prob = TRUE, add = TRUE)
expectedValueMTE(origF)
mean(X)
```

---

rescale_data *Scale data*

Description

Standardize the numeric columns of a data.frame.

Usage

```
rescale_data(data, v_mean = NULL, v_sd = NULL)
```

Arguments

data	data.frame containing the variables to be standardized.
v_mean	vector of means of the data.frame. The value for the non-numeric columns should be NA. If not given, it is computed from data.
v_sd	vector of standard deviations of the data.frame. The value for the non-numeric columns should be NA. If not given, it is computed from data.

Value

A data.frame with scaled data. Non-numeric columns are also returned in their original position.

---

rnormMultiv *Multivariate Normal sampling*

Description

Generate a multivariate normal data vector taking into account the real data and the relationships with other variables in the dataset.

Usage

```
rnormMultiv(n, dataParents, dataChild)
```

Arguments

n	A "numeric" value which is the size of the prior data to generate.
dataParents	A data set of class "data.frame" giving the data of the set of coditional parent variables.
dataChild	A "numeric" vector containing the original data of the child variable.

Value

A "numeric" vector giving the prior data values.

See Also

generateNormalPriorData

Examples

```
## Data
data(ecoli)
data <- ecoli[,-c(1,9)] ## remove sequece.name and class

## DAG
dag <- LearningHC(data)
plot(dag)
```

```
getChildParentsFromGraph(dag)

## 1. Random sample
parents <- "mcg"
child <- "alm1"
n <- 1000
rnormMultiv(n, dataParents = data.frame(data[,parents]), dataChild = data[,child])

## 2. Random sample
parents <- "alm1"
child <- "aac"
n <- 256
rnormMultiv(n, dataParents = data.frame(data[,parents]), dataChild = data[,child])
```

`sample_motbfs` *Generate Samples From an MoTBF Bayesian network*

Description

This function generates a sample from an MoTBF Bayesian network.

Usage

```
sample_motbfs(bn, n, parallel = FALSE, evidence = NULL)
```

Arguments

`bn`	An object of class motbf_fit, obtained from the function motbf.fit.
`n`	A non-negative integer giving the number of instances to be generated.
`parallel`	A `logical` value. If TRUE, parallelization is carried out. As a default, it is set to FALSE
`evidence`	A data.frame of one row containing the values for the observed variables. As a default, it is NULL.

Value

A `data.frame` containing the generated sample. Is evidence is not NULL, attribute 'w' contains the weight of each sample.

Examples

```
data("ecoli", package = "MoTBFs")
dat <- ecoli[,-c(1,4,5,9)]

# Build DAG
dag <- LearningHC(dat)
```

```
 # Learn BN parameters
 bn = motbf.fit(dag, dat)

# Get sample from bn
sam = sample_motbfs(bn, 50)
```

---

subsetData *Dataset subsetting*

Description

Collection of functions for subsetting a "data.frame" by rows or columns, and to create training and test partitions.

Usage

```
splitFolds(data, k)

TrainingandTestData(data, percentage_test, discreteVariables = NULL)

splitdata(data, nameVariable, min, max)
```

Arguments

data	A dataset of class data.frame.
k	The number of folds for k-fold cross validation, used in splitFolds function.
percentage_test	The proportion of data that goes to the test set (between 0 and 1), used in TrainingandTestData function.
discreteVariables	A character vector with the name of the discrete variables in the dataset.
nameVariable	A character vector with the name of the variable to be filtered, used in splitdata function.
min, max	Boundary values to filter out, used in splitdata function.

Value

TrainingandTestData() returns a list of 2 elements containing the train and test datasets. splitdata() returns a subset of observations.

Examples

```
## Dataset
X <- rnorm(1000)
Y <- rchisq(1000, df = 8)
Z <- rep(letters[1:10], times = 1000/10)
data <- data.frame(X = X, Y = Y, Z = Z)

## Training and Test Datasets
TT <- TrainingandTestData(data, percentage_test = 0.2)
TT$Training
TT$Test

## Subset Dataset
splitdata(data, nameVariable = "X", min = 2, max= 3)
```

---

summary.motbf *Summarize an "motbf" object by describing its main features.*

Description

Summarize an "motbf" object by describing its main features.

Usage

```
## S3 method for class 'motbf'
summary(object, ...)

## S3 method for class 'motbf_fit_cv'
summary(object, ...)

## S3 method for class 'univmotbf'
summary(object, ...)

## S3 method for class 'summary.univmotbf'
print(x, ...)

## S3 method for class 'piecewisemop'
summary(object, ...)

## S3 method for class 'summary.piecewisemop'
print(x, ...)

## S3 method for class 'jointmotbf'
summary(object, ...)

## S3 method for class 'summary.jointmotbf'
print(x, ...)
```

Arguments

object	An object of class "motbf".
...	further arguments passed to or from other methods.
x	An object of class "summary.motbf".

Value

The summary of an "motbf" object. It contains a list of elements with the most important information of the object.

See Also

univMoTBF

Examples

```
## Subclass 'MOP'
X <- rnorm(1000)
P <- univMoTBF(X, POTENTIAL_TYPE="MOP") ## or POTENTIAL_TYPE="MTE"
summary(P)
attributes(sP <- summary(P))
attributes(sP)
sP$Function
sP$Subclass
sP$Iterations

## Subclass 'MTE'
X <- rnorm(1000)
P <- univMoTBF(X, POTENTIAL_TYPE="MTE")
summary(P)
attributes(sP <- summary(P))
attributes(sP)
sP$Function
sP$Subclass
sP$Iterations
```

---

thyroid *Data set Thyroid Disease (thyroid0387)*

Description

This data set if one of the several databases about Thyroid avalaible at the UCI repository. The task is to detect is a given patient is normal (1) or suffers from hyperthyroidism (2) or hypothyroidism (3) .

Format

A data frame with 7200 rows, 21 variables and the class.

Details

Age Age of the patient (0.01–0.97). Continuous variable.

Sex Sex of the patient, 0 (Male) 1 (Female). Binary variable.

On_thyroxine 0 (FALSE) 1 (TRUE). Binary variable.

Query_on_thyroxine 0 (FALSE) 1 (TRUE). Binary variable.

On_antithyroid_medication 0 (FALSE) 1 (TRUE). Binary variable.

Sick 0 (FALSE) 1 (TRUE). Binary variable.

Pregnant 0 (FALSE) 1 (TRUE). Binary variable.

Thyroid_surgery 0 (FALSE) 1 (TRUE). Binary variable.

I131_treatment 0 (FALSE) 1 (TRUE). Binary variable.

Query_hypothyroid 0 (FALSE) 1 (TRUE). Binary variable.

Query_hyperthyroid 0 (FALSE) 1 (TRUE). Binary variable.

Lithium 0 (FALSE) 1 (TRUE). Binary variable.

Goitre 0 (FALSE) 1 (TRUE). Binary variable.

Tumor 0 (FALSE) 1 (TRUE). Binary variable.

Hypopituitary 0 (FALSE) 1 (TRUE). Binary variable.

Psych 0 (FALSE) 1 (TRUE). Binary variable.

TSH amount of TSH (0.0–0.53). Continuous variable.

T3 amount of T3 (0.0005–0.18). Continuous variable.

TT4 amount of TT4 (0.002–0.6). Continuous variable.

T4U amount of T4U (0.017–0.233). Continuous variable.

FTI amount of FTI (0.002–0.642). Continuous variable.

Class 1 (normal) 2 (hyperthyroidism) 3 (hypothyroidism). Class variable.

Source

http://archive.ics.uci.edu/ml/datasets/Thyroid+Disease

---

univMoTBF *Fitting MoTBFs*

Description

Function for fitting univariate mixture of truncated basis functions. Least square optimization is used to minimize the quadratic error between the empirical cumulative distribution and the estimated one.

Usage

```
univMoTBF(
  x,
  POTENTIAL_TYPE,
  evalRange = NULL,
  nparam = NULL,
  maxParam = NULL,
  scale = TRUE
)
```

Arguments

x	A "numeric" vector.
POTENTIAL_TYPE	A "character" string specifying the potential type, must be either "MOP" or "MTE".
evalRange	A "numeric" vector that specifies the domain over which the model will be fitted. By default, it is NULL and the function is defined over the complete data range.
nparam	The exact number of basis functions to be used. By default, it is NULL and the best MoTBF is fitted taking into account the Bayesian information criterion (BIC) to score and select the functions. It evaluates the next two functions and, if the BIC value does not improve, the function with the best BIC score so far is returned.
maxParam	A "numeric" value which indicates the maximum number of coefficients in the function. By default, it is NULL; otherwise, the function which gets the best BIC score with at most this number of parameters is returned.
scale	A "logical" value indicating whether to standardize the numeric vector (x) to have mean 0 and standard deviation 1.

Value

univMoTBF() returns an object of class "motbf". This object is a list containing several elements, including its mathematical expression and other hidden elements related to the learning task. The processing time is one of the values returned by this function and it can be extracted by $Time. Although the learning process is always the same for a particular data sample, the processing can vary inasmuch as it depends on the CPU.

Examples

```
## 1. EXAMPLE
## Data
X <- rnorm(5000)

## Learning
f1 <- univMoTBF(X, POTENTIAL_TYPE = "MTE"); f1
f2 <- univMoTBF(X, POTENTIAL_TYPE = "MOP"); f2

## Plots
```

```
hist(X, prob = TRUE, main = "")
plot(f1, xlim = range(X), col = 1, add = TRUE)
plot(f2, xlim = range(X), col = 2, add = TRUE)

## Data test
Xtest <- rnorm(1000)
## Filtered data test
Xtest <- Xtest[Xtest>=min(X) & Xtest<=max(X)]

## Log-likelihood
sum(log(as.function(f1)(Xtest)))
sum(log(as.function(f2)(Xtest)))

## 2. EXAMPLE
## Data
X <- rchisq(5000, df = 5)

## Learning
f1 <- univMoTBF(X, POTENTIAL_TYPE = "MTE", nparam = 11); f1
f2 <- univMoTBF(X, POTENTIAL_TYPE = "MOP", maxParam = 10); f2

## Plots
hist(X, prob = TRUE, main = "")
plot(f1, xlim = range(X), col = 3, add = TRUE)
plot(f2, xlim = range(X), col = 4, add = TRUE)

## Data test
Xtest <- rchisq(1000, df = 5)
## Filtered data test
Xtest <- Xtest[Xtest>=min(X) & Xtest<=max(X)]

## Log-likelihood
sum(log(as.function(f1)(Xtest)))
sum(log(as.function(f2)(Xtest)))
```

`UpperBoundLogLikelihood` *Upper bound of the loglikelihood*

Description

Computes an upper bound of the expected loglikelihood of a dataset given a randomly generated MoTBF density.

Usage

```
UpperBoundLogLikelihood(f, data, min, max)
```

Arguments

f	A function to evaluate of class "character", "motbf" or others.
data	A "numeric" array which contains the values to evaluate.
min	A "numeric" value giving the lower limit of the domain.
max	A "numeric" value giving the upper limit of the domain.

Value

A "numeric" value which is the log-likelihood of the evaluated ramdom function.

See Also

getNonNormalisedRandomMoTBF

Examples

```
data <- rnorm(20)
f <- getNonNormalisedRandomMoTBF(degree = 8, POTENTIAL_TYPE = "MOP")
UpperBoundLogLikelihood(f, data, min = -2.5, max = 3.2)

data <- rexp(20)
f <- getNonNormalisedRandomMoTBF(degree = 8, POTENTIAL_TYPE = "MTE")
UpperBoundLogLikelihood(f, data, min = 0, max = 5)
```

---

variableElimination *Exact inference*

Description

Compute the posterior distribution of a variable of interest given some evidence. The variable elimination algorithm is used.

Usage

```
variableElimination(bn, target, evidence = NULL, elimOrder = NULL)
```

Arguments

bn	An object of class motbf_fit, obtained from function motbf.fit.
target	A character string equal to the name of the variable of interest.
evidence	A data.frame of one row containing the value of the observed variables. A list can also be provided.
elimOrder	The elimination order can be manually specified as a vector containing the names of the variables, in the desired order. If elimOrder is not specified, the topological order is computed.

Value

The posterior probability distribution of the target variable as an object of class `univmotbf` or `piecewisemop` if `target` is continuous, or a matrix if `target` is discrete.

Examples

```
## Dataset
  data("ecoli", package = "MoTBFs")
  data <- ecoli[,-c(1,9)]

## Get directed acyclic graph
  dag <- LearningHC(data)

## Learn bayesian network
  bn <- motbf.fit(dag, data = data, numIntervals = 4, POTENTIAL_TYPE = "MOP")

## Specify the evidence set and target variable
  obs <- data.frame(lip = "0.48", alm1 = 0.55, stringsAsFactors=FALSE)
  node <- "alm2"
ve = variableElimination(bn, target = node, evidence = obs)
```

---

variableSelection *Variable selection for MoTBFs*

Description

Perform variable selection for Bayesian networks of class MoTBF.

Usage

```
variableSelection(
  data,
  dag,
  loss,
  target = NULL,
  order = NULL,
  method = "forward",
  fit.args = NULL,
  loss.args = NULL,
  cv.args = NULL,
  verbose = TRUE
)
```

Arguments

`data`	an object of class `"data.frame"`, which can contain continuous and discrete variables.

dag	a character string indicating the structural learning algorithm to be applied to the training data. Available options are naive Bayes (NB), tree augmented naive Bayes (TAN) and hill-climbing (HC).
loss	a character string indicating which loss function should be used. Currently, two options are available: 'logl', for the log-likelihood of the model; and 'pred', for the predictive error. See details.
target	a character string indicating which node is the target.
order	a character vector indicating the order in which the predictive variables are included in the model. If it is NULL, the predictors are ordered regarding their mutual information with the target variable.
method	a character vector indicating the method used for the variable selection. Available methods are 'forward' (default), 'gf' (greedy forward selection) and 'iwsr' (Incremental Wrapper Sequential Subset with Replacement).
fit.args	a list containing optional arguments used to fit the models. These arguments must be those accepted by function motbf.fit, i.e., 'numIntervals' (4), 'POTENTIAL_TYPE' ('MOP'), 'maxParam' (NULL), 's' (NULL), 'priorData' (NULL) or 'scale' (TRUE). If fit.args is left NULL, the default values (in brackets) for those arguments will be used.
loss.args	a list containing optional arguments related to the loss functions. Currently available arguments are: 'loss.matrix' and 'percentage_test'. See details.
cv.args	a list containing optional arguments related to the cross-validation method. Currently available arguments are: 'k', 'seed'. See details.
verbose	Logical; if TRUE, prints execution messages and progress updates to the console.

Details

A filter-wrapper variable selection procedure is implemented. Firstly, the explanatory variables are ordered according to their mutual information with the target variable, unless argument 'order' is not null, in which case the given order is followed. Then, the first variable in the ordered set and the target are used to fit an initial model. The model is validated by means of a k-fold cross validation (cv.args[[k]] >= 2). However, it is possible to validate on a test set (cv.args[[k]] = 1), or on the same training set (cv.args[[k]] = 0). The model is evaluated in terms of its log-likelihood (loss = 'logl') or its predictive accuracy (loss = 'pred'). In the latter case, the root mean squared error is computed for regression models, whereas the classification accuracy is computed for classification models. Afterwards, the remaining explanatory variables are included in the model, one by one, according to the aforementioned order, and a new model is obtained. Whenever the inclusion of a variable increases the accuracy of the model (increases its log-likelihood or classification error, or decreases its root mean squared error), it is kept; otherwise, it is excluded from the model.

Details on the loss argument:

'logl' The log-likelihood of the model is computed. This measure is available for both basis functions, MTE and MOP.

'pred' This option is only available for MOPs. The predictive error (root mean squared error) or classification accuracy is computed as the loss function, depending on the nature of the target variable (continuous or discrete, respectively). The program will guess which type the target variable is and compute the corresponding measure.

Details on the `loss.args` argument. Currently, two arguments can be specified within this list:

`'p.test'` Only used if `k = 1`. This argument specifies the proportion of the data set that goes to the test set (between 0 and 1).

`'loss.matrix'` A squared `matrix` used to compute a weighted classification accuracy for discrete targets. This matrix is multiplied by the confusion matrix element by element, and the resulting matrix is used to compute the classification accuracy. The loss matrix allows to increase the penalty of some user-specified errors. If the loss matrix provided is a constant matrix of ones, the result is the standard classification accuracy. Note that the diagonal of the loss matrix is regarded as a reward, while the off-diagonal is regarded as a cost.

Details in the `cv.args` argument. Currently, two arguments can be specified within this list:

`k` an integer indicating the number of folds to split the data set. If k = 0, the train and test sets are the same data; if k = 1, hold-out validation is carried out, i.e., the data set is split in train (80% by default) and test (20% by default); finally, if k >=2, k-fold cross validation is carried out.

`seed` an integer to specify the seed. The k-folds are created randomly, so one might expect slightly different results unless 'seed' is used.

Value

A list of 4 elements

data a `"data.frame"` of the selected variables, including the target.

index the index of the selected variables (with respect to the input dataset), including the target.

loss the loss value of the best model.

crossvalidation an object of class `"motbf.fit.cv"`, containing the result of the cross-validation of the best model.

Examples

```
#################
### EXAMPLE 1 ###
#################
# Perform variable selection on the iris dataset.
# Use the TAN structure as DAG and the Species variable as target.
vs = variableSelection(iris, dag = 'TAN', loss = 'pred', target = 'Species',
  cv.args = list(k = 1, seed = 1023))

# Check out the results of the best model.
summary(vs$crossvalidation)
```

Index