Package 'dagHMM'

July 18, 2025

Type Package

Title Directed Acyclic Graph HMM with TAN Structured Emissions

Version 0.1.1

Maintainer Prajwal Bende <prajwal.bende@gmail.com>

Description Hidden Markov models (HMMs) are a formal foundation for making probabilistic models of linear sequence. They provide a conceptual toolkit for building complex models just by drawing an intuitive picture. They are at the heart of a diverse range of programs, including genefinding, profile searches, multiple sequence alignment and regulatory site identification. HMMs are the Legos of computational sequence analysis. In graph theory, a tree is an undirected graph in which any two vertices are connected by exactly one path, or equivalently a connected acyclic undirected graph. Tree represents the nodes connected by edges. It is a non-linear data structure. A poly-tree is simply a directed acyclic graph whose underlying undirected graph is a tree. The model proposed in this package is the same as an HMM but where the states are linked via a polytree structure rather than a simple path.

License GPL (>= 2.0.0)

Encoding UTF-8

Imports gtools, future, matrixStats, PRROC, bnlearn, bnclassify

RoxygenNote 7.2.3

NeedsCompilation no

Author Prajwal Bende [aut, cre],
Russ Greiner [ths],
Pouria Ramazi [ths]

Repository CRAN

Date/Publication 2025-07-18 15:20:29 UTC

Contents

`backward` *Infer the backward probabilities for all the nodes of the dagHMM*

Description

`backward` calculates the backward probabilities for all the nodes

Usage

```
backward(hmm, observation, bt_seq, kn_states = NULL)
```

Arguments

`hmm`	hmm Object of class List given as output by `initHMM`
`observation`	Dataframe containing the discritized character values of only covariates at each node. Column names of dataframe should be same as the covariate names. Missing values should be denoted by "NA".
`bt_seq`	A vector denoting the order of nodes in which the DAG should be traversed in backward direction(from leaves to roots). Output of `bwd_seq_gen` function.
`kn_states`	(Optional) A (L * 2) dataframe where L is the number of training nodes where state values are known. First column should be the node number and the second column being the corresponding known state values of the nodes

Details

The backward probability for state X and observation at node k is defined as the probability of observing the sequence of observations e_k+1, ... ,e_n under the condition that the state at node k is X. That is:
`b[X,k] := Prob(E_k+1 = e_k+1, ... , E_n = e_n | X_k = X)`
where `E_1...E_n = e_1...e_n` is the sequence of observed emissions and `X_k` is a random variable that represents the state at node k

Value

(2 * N) matrix denoting the backward probabilites at each node of the dag, where "N" is the total number of nodes in the dag

See Also

forward

Examples

```
library(bnlearn)

tmat = matrix(c(0,0,1,0,0,0,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0),
              5,5, byrow= TRUE ) #for "X" (5 nodes) shaped dag
states = c("P","N") #"P" represent cases(or positive) and "N" represent controls(or negative)
bnet = model2network("[A][C|A:B][D|A:C][B|A]") #A is the target variable while
                                               #B, C and D are covariates
obsvA=data.frame(list(B=c("L","H","H","L","L"),C=c("H","H","L","L","H"),D=c("L","L","L","H","H")))
hmmA = initHMM(States=states, dagmat= tmat, net=bnet, observation=obsvA)
bt_sq = bwd_seq_gen(hmmA)
kn_st = data.frame(node=c(3),state=c("P"),stringsAsFactors = FALSE)
                   #state at node 3 is known to be "P"
BackwardProbs = backward(hmm=hmmA,observation=obsvA,bt_seq=bt_sq,kn_states=kn_st)
```

---

baumWelch *Inferring the parameters of a dag Hidden Markov Model via the Baum-Welch algorithm*

Description

For an initial Hidden Markov Model (HMM) with some assumed initial parameters and a given set of observations at all the nodes of the dag, the Baum-Welch algorithm infers optimal parameters to the HMM. Since the Baum-Welch algorithm is a variant of the Expectation-Maximisation algorithm, the algorithm converges to a local solution which might not be the global optimum. Note that if you give the training and validation data, the function will message out AUC and AUPR values after every iteration. Also, validation data must contain more than one instance of either of the possible states

Usage

```
baumWelch(
  hmm,
  observation,
  kn_states = NULL,
  kn_verify = NULL,
  maxIterations = 50,
  delta = 1e-05,
  pseudoCount = 1e-100
)
```

Arguments

hmm	hmm Object of class List given as output by initHMM
observation	Dataframe containing the discritized character values of only covariates at each node. Column names of dataframe should be same as the covariate names. Missing values should be denoted by "NA".
kn_states	(Optional) A (L * 2) dataframe where L is the number of training nodes where state values are known. First column should be the node number and the second column being the corresponding known state values of the nodes
kn_verify	(Optional) A (L * 2) dataframe where L is the number of validation nodes where state values are known. First column should be the node number and the second column being the corresponding known state values of the nodes
maxIterations	(Optional) The maximum number of iterations in the Baum-Welch algorithm. Default is 50
delta	(Optional) Additional termination condition, if the transition and emission matrices converge, before reaching the maximum number of iterations (maxIterations). The difference of transition and emission parameters in consecutive iterations must be smaller than delta to terminate the algorithm. Default is 1e-5
pseudoCount	(Optional) Adding this amount of pseudo counts in the estimation-step of the Baum-Welch algorithm. Default is 1e-100 (Don't keep it zero to avoid numerical errors)

Value

List of three elements, first being the infered HMM whose representation is equivalent to the representation in initHMM, second being a list of statistics of algorithm and third being the final state probability distribution at all nodes.

See Also

baumWelchRecursion

Examples

```
library(bnlearn)

tmat = matrix(c(0,0,1,0,0,0,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0),
              5,5, byrow= TRUE ) #for "X" (5 nodes) shaped dag
states = c("P","N") #"P" represent cases(or positive) and "N" represent controls(or negative)
bnet = model2network("[A][C|A:B][D|A:C][B|A]") #A is the target variable while
                                               #B, C and D are covariates.
obsvA=data.frame(list(B=c("L","H","H","L","L"),C=c("H","H","L","L","H"),D=c("L","L","L","H","H")))
hmmA = initHMM(States=states, dagmat= tmat, net=bnet, observation=obsvA)
kn_st = data.frame(node=c(2),state=c("P"),stringsAsFactors = FALSE)
                  #state at node 2 is known to be "P"
kn_vr = data.frame(node=c(3,4,5),state=c("P","N","P"),stringsAsFactors = FALSE)
                  #state at node 3,4,5 are "P","N","P" respectively
learntHMM= baumWelch(hmm=hmmA,observation=obsvA,kn_states=kn_st, kn_verify=kn_vr)
```

baumWelchRecursion *Implementation of the Baum Welch Algorithm as a special case of EM algorithm*

Description

baumWelch recursively calls this function to give a final estimate of parameters for dag HMM Uses Parallel Processing to speed up calculations for large data. Should not be used directly.

Usage

```
baumWelchRecursion(hmm, observation, t_seq, kn_states = NULL, kn_verify = NULL)
```

Arguments

hmm	hmm Object of class List given as output by initHMM
observation	Dataframe containing the discritized character values of only covariates at each node. Column names of dataframe should be same as the covariate names. Missing values should be denoted by "NA".
t_seq	list of forward and backward DAG traversal sequence (in that order) as given output by fwd_seq_gen and bwd_seq_gen function
kn_states	(Optional) A (L * 2) dataframe where L is the number of training nodes where state values are known. First column should be the node number and the second column being the corresponding known state values of the nodes
kn_verify	(Optional) A (L * 2) dataframe where L is the number of validation nodes where state values are known. First column should be the node number and the second column being the corresponding known state values of the nodes

Value

List containing estimated Transition and Emission probability matrices

See Also

baumWelch

Examples

```
library(bnlearn)

tmat = matrix(c(0,0,1,0,0,0,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0),
              5,5, byrow= TRUE ) #for "X" (5 nodes) shaped dag
states = c("P","N") #"P" represent cases(or positive) and "N" represent controls(or negative)
bnet = model2network("[A][C|A:B][D|A:C][B|A]") #A is the target variable while
                                               #B, C and D are covariates
obsvA=data.frame(list(B=c("L","H","H","L","L"),C=c("H","H","L","L","H"),D=c("L","L","L","H","H")))
hmmA = initHMM(States=states, dagmat= tmat, net=bnet, observation=obsvA)
```

```
kn_st = data.frame(node=c(2),state=c("P"),stringsAsFactors = FALSE)
                  #state at node 2 is known to be "P"
kn_vr = data.frame(node=c(3,4,5),state=c("P","N","P"),stringsAsFactors = FALSE)
                  #state at node 3,4,5 are "P","N","P" respectively
t_seq=list(fwd_seq_gen(hmmA),bwd_seq_gen(hmmA))
newparam= baumWelchRecursion(hmm=hmmA,observation=obsvA,t_seq=t_seq,
                             kn_states=kn_st, kn_verify=kn_vr)
```

---

bwd_seq_gen *Calculate the order in which nodes in the dag should be traversed during the backward pass(leaves to roots)*

Description

dag is a complex graphical model where we can have multiple parents and multiple children for a node. Hence the order in which the dag should be tranversed becomes significant. Backward algorithm is a dynamic programming problem where to calculate the values at a node, we need the values of the children nodes beforehand, which need to be traversed before this node. This algorithm outputs a possible(not unique) order of the traversal of nodes ensuring that the childrens are traversed first before the parents

Usage

```
bwd_seq_gen(hmm, nlevel = 100)
```

Arguments

hmm	hmm Object of class List given as output by initHMM
nlevel	No. of levels in the dag, if known. Default is 100

Value

Vector of length "D", where "D" is the number of nodes in the dag

See Also

backward

Examples

```
library(bnlearn)

tmat = matrix(c(0,0,1,0,0,0,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0),
              5,5, byrow= TRUE ) #for "X" (5 nodes) shaped dag
states = c("P","N") #"P" represent cases(or positive) and "N" represent controls(or negative)
bnet = model2network("[A][C|A:B][D|A:C][B|A]") #A is the target variable while
                                               #B, C and D are covariates
obsvA=data.frame(list(B=c("L","H","H","L","L"),C=c("H","H","L","L","H"),D=c("L","L","L","H","H")))
hmmA = initHMM(States=states, dagmat= tmat, net=bnet, observation=obsvA)
bt_sq = bwd_seq_gen(hmmA)
```

`calc_emis` *Calculating the probability of occurance of particular values of covariates at a node given the value of target.*

Description

Calculating the probability of occurance of particular values of covariates at a node given the value of target.

Usage

```
calc_emis(state, obsv, probs, pars)
```

Arguments

`state`	character value of state variable at a particular node.
`obsv`	character vector of values of covariates at that node.
`probs`	emission probability distribution of the covariates in TAN structure.
`pars`	integer vector denoting the parents of the nodes(other than root) in the TAN structure.

Value

probability of occurance of particular values of covariates at a node given the value of target.

`forward` *Infer the forward probabilities for all the nodes of the dagHMM*

Description

`forward` calculates the forward probabilities for all the nodes

Usage

```
forward(hmm, observation, ft_seq, kn_states = NULL)
```

Arguments

`hmm`	hmm Object of class List given as output by `initHMM`
`observation`	Dataframe containing the discritized character values of only covariates at each node. Column names of dataframe should be same as the covariate names. Missing values should be denoted by "NA".
`ft_seq`	A vector denoting the order of nodes in which the DAG should be traversed in forward direction(from roots to leaves). Output of `fwd_seq_gen` function.
`kn_states`	(Optional) A (L * 2) dataframe where L is the number of training nodes where state values are known. First column should be the node number and the second column being the corresponding known state values of the nodes

Details

The forward probability for state X up to observation at node k is defined as the probability of observing the sequence of observations e_1,..,e_k given that the state at node k is X. That is:
`f[X,k] := Prob( X_k = X | E_1 = e_1, .., E_k = e_k)`
where `E_1...E_n = e_1...e_n` is the sequence of observed emissions and `X_k` is a random variable that represents the state at node k

Value

(2 * N) matrix denoting the forward probabilites at each node of the dag, where "N" is the total number of nodes in the dag

See Also

backward

Examples

```
library(bnlearn)

tmat = matrix(c(0,0,1,0,0,0,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0),
              5,5, byrow= TRUE ) #for "X" (5 nodes) shaped dag
states = c("P","N") #"P" represent cases(or positive) and "N" represent controls(or negative)
bnet = model2network("[A][C|A:B][D|A:C][B|A]") #A is the target variable while
                                               #B, C and D are covariates
obsvA=data.frame(list(B=c("L","H","H","L","L"),C=c("H","H","L","L","H"),D=c("L","L","L","H","H")))
hmmA = initHMM(States=states, dagmat= tmat, net=bnet, observation=obsvA)
ft_sq = fwd_seq_gen(hmmA)
kn_st = data.frame(node=c(3),state=c("P"),stringsAsFactors = FALSE)
                  #state at node 3 is known to be "P"
ForwardProbs = forward(hmm=hmmA,observation=obsvA,ft_seq=ft_sq,kn_states=kn_st)
```

`fwd_seq_gen` *Calculate the order in which nodes in the dag should be traversed during the forward pass(roots to leaves)*

Description

dag is a complex graphical model where we can have multiple parents and multiple children for a node. Hence the order in which the dag should be tranversed becomes significant. Forward algorithm is a dynamic programming problem where to calculate the values at a node, we need the values of the parent nodes beforehand, which need to be traversed before this node. This algorithm outputs a possible(not unique) order of the traversal of nodes ensuring that the parents are traversed first before the children.

Usage

```
fwd_seq_gen(hmm, nlevel = 100)
```

Arguments

hmm	hmm Object of class List given as output by initHMM
nlevel	No. of levels in the dag, if known. Default is 100

Value

Vector of length "D", where "D" is the number of nodes in the dag

See Also

forward

Examples

```
library(bnlearn)

tmat = matrix(c(0,0,1,0,0,0,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0),
              5,5, byrow= TRUE ) #for "X" (5 nodes) shaped dag
states = c("P","N") #"P" represent cases(or positive) and "N" represent controls(or negative)
bnet = model2network("[A][C|A:B][D|A:C][B|A]") #A is the target variable while
                                               #B, C and D are covariates
obsvA=data.frame(list(B=c("L","H","H","L","L"),C=c("H","H","L","L","H"),D=c("L","L","L","H","H")))
hmmA = initHMM(States=states, dagmat= tmat, net=bnet, observation=obsvA)
ft_sq = fwd_seq_gen(hmmA)
```

---

gen_emis *Generating the inital emission probability distribution of the covariates in TAN structure.*

Description

Generating the inital emission probability distribution of the covariates in TAN structure.

Usage

```
gen_emis(net, observation, sym)
```

Arguments

net	Object of type 'bn' provided as output by model2network showing the TAN structure between target variable and covariates.
observation	Dataframe containing the discritized character values of only covariates at each node. Column names of dataframe should be same as the covariate names. Missing values should be denoted by "NA".
sym	Character vector of possible values of target variable

Value

Inital emission probability distribution of the covariates in TAN structure

Examples

```
library(bnlearn)

bnet = model2network("[A][C|A:B][D|A:C][B|A]") #A is the target variable while
                                               #B, C and D are covariates
obsvA=data.frame(list(B=c("L","H","H","L","L"),C=c("H","H","L","L","H"),D=c("L","L","L","H","H")))
target_value= c("P","N")
prob= gen_emis(net=bnet,observation=obsvA,sym=target_value)
```

`initHMM` *Initializing dagHMM with given parameters*

Description

Initializing dagHMM with given parameters

Usage

```
initHMM(
  States,
  dagmat,
  net = NULL,
  observation,
  startProbs = NULL,
  transProbs = NULL,
  leak_param = 0
)
```

Arguments

`States`	A (2 * 1) vector with first element being discrete state value for the cases(or positive) and second element being discrete state value for the controls(or negative) for given dagHMM
`dagmat`	Adjacent Symmetry Matrix that describes the topology of the dag
`net`	Object of type 'bn' provided as output by model2network showing the TAN structure between target variable and covariates.
`observation`	Dataframe containing the discritized character values of covariates at each node. If "net" is not given, dataframe should also contain the column for target variable (as the last column) so as to learn the structure. Column names of dataframe should be same as the covariate names. Missing values should be denoted by "NA".

startProbs	(Optional) (2 * 1) vector containing starting probabilities for the states. Default is equally probable states
transProbs	(Optional) (2 * 2) matrix containing transition probabilities for the states.
leak_param	(Optional) Leak parameter used in Noisy-OR algorithm used in forward and noisy_or.Default is 0

Value

List describing the parameters of dagHMM(pi, alpha, beta, dagmat, net)

Examples

```
library(bnlearn)

tmat = matrix(c(0,0,1,0,0,0,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0),
              5,5, byrow= TRUE ) #for "X" (5 nodes) shaped dag
states = c("P","N") #"P" represent cases(or positive) and "N" represent controls(or negative)
bnet = model2network("[A][C|A:B][D|A:C][B|A]") #A is the target variable while
                                               #B, C and D are covariates.
obsvA=data.frame(list(B=c("L","H","H","L","L"),C=c("H","H","L","L","H"),D=c("L","L","L","H","H")))
hmmA = initHMM(States=states, dagmat= tmat, net=bnet, observation=obsvA)
obsvB=data.frame(list(B=c("L","H","H","L","L"),C=c("H","H","L","L","H"),D=c("L","L","L","H","H")),
                      A=c("P","N","P","P","N"))
hmmB = initHMM(States=states, dagmat= tmat, net=NULL, observation=obsvB)
```

---

noisy_or *Calculating the probability of transition from multiple nodes to given node in the dag*

Description

Calculating the probability of transition from multiple nodes to given node in the dag

Usage

```
noisy_or(hmm, prev_state, cur_state)
```

Arguments

hmm	Object of class List given as output by initHMM,
prev_state	vector containing state variable values for the previous nodes
cur_state	character denoting the state variable value for current node

Value

The Noisy_OR probability for the transition

Examples

```
library(bnlearn)

tmat = matrix(c(0,0,1,0,0,0,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0),
               5,5, byrow= TRUE ) #for "X" (5 nodes) shaped dag
states = c("P","N") #"P" represent cases(or positive) and "N" represent controls(or negative)
bnet = model2network("[A][C|A:B][D|A:C][B|A]") #A is the target variable while
                                               #B, C and D are covariates.
obsvA=data.frame(list(B=c("L","H","H","L","L"),C=c("H","H","L","L","H"),D=c("L","L","L","H","H")))
hmmA = initHMM(States=states, dagmat= tmat, net=bnet, observation=obsvA)
Transprob = noisy_or(hmm=hmmA,prev_state=c("P","N"),cur_state="P") #for transition from P & N
                                                                   #simultaneously to P
```

Index