Package 'treeHMM'

October 13, 2019

Type Package

Title Tree Structured Hidden Markov Model

Version 0.1.0

Author Prajwal Bende [aut, cre]

Maintainer Prajwal Bende <pbende@ualberta.ca>

Description Used for Inference, Prediction and Parameter learning for tree structured Hidden Markov Model. The package propose a new architecture of Hidden Markov Model(HMM) known as Tree Structured HMM which could be used in various applications which involves graphs, trees etc.

License GPL (>= 2.0.0)

Encoding UTF-8

LazyData true

Imports Matrix, gtools, future, matrixStats, PRROC

RoxygenNote 6.1.1

NeedsCompilation no

Repository CRAN

Date/Publication 2019-10-13 11:10:02 UTC

R topics documented:

backward *Infer the backward probabilities for all the nodes of the treeHMM*

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	A list consisting "k" vectors for "k" features, each vector being a character series of discrete emmision values at different nodes serially sorted by node number
bt_seq	A vector denoting the order of nodes in which the tree 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

(N * D) matrix denoting the backward probabilites at each node of the tree, where "N" is possible no. of states and "D" is the total number of nodes in the tree

See Also

forward

Examples

```
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 tree
hmmA = initHMM(c("P","N"),list(c("L","R")), tmat) #one feature with two discrete levels "L" and "R"
obsv = list(c("L","L","R","R","L")) #emissions for the one feature for the 5 nodes in order 1:5
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(hmmA,obsv,bt_sq,kn_st)
```

`baumWelch` *Inferring the parameters of a tree 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 tree, 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 = 0)
```

Arguments

`hmm`	hmm Object of class List given as output by `initHMM`
`observation`	A list consisting "k" vectors for "k" features, each vector being a character series of discrete emmision values at different nodes serially sorted by node number
`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 100
`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-9
`pseudoCount`	(Optional) Adding this amount of pseudo counts in the estimation-step of the Baum-Welch algorithm. Default is zero

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

```
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 tree
hmmA= initHMM(c("P","N"),list(c("L","R")), tmat) #one feature with two discrete levels "L" and "R"
obsv= list(c("L","L","R","R","L")) #emissions for the one feature for the 5 nodes in order 1:5
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(hmmA,obsv,kn_st, 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 tree HMM Uses Parallel Processing to speed up calculations for large data. Should not be used directly.

Usage

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

Arguments

hmm	hmm Object of class List given as output by initHMM
observation	A list consisting "k" vectors for "k" features, each vector being a character series of discrete emmision values at different nodes serially sorted by node number
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

```
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 tree
hmmA = initHMM(c("P","N"),list(c("L","R")), tmat) #one feature with two discrete levels "L" and "R"
obsv = list(c("L","L","R","R","L")) #emissions for the one feature for the 5 nodes in order 1:5
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
newparam= baumWelchRecursion(hmmA,obsv,kn_st, kn_vr)
```

---

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

Description

Tree is a complex graphical model where we can have multiple parents and multiple children for a node. Hence the order in which the tree 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 tree, if known. Default is 100

Value

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

See Also

backward

Examples

```
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 tree
hmmA = initHMM(c("A","B"),list(c("L","R")), tmat) #one feature with two discrete levels "L" and "R"
bt_sq = bwd_seq_gen(hmmA)
```

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

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`	A list consisting "k" vectors for "k" features, each vector being a character series of discrete emmision values at different nodes serially sorted by node number
`ft_seq`	A vector denoting the order of nodes in which the tree 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

(N * D) matrix denoting the forward probabilites at each node of the tree, where "N" is possible no. of states and "D" is the total number of nodes in the tree

See Also

`backward`

Examples

```
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 tree
hmmA = initHMM(c("P","N"),list(c("L","R")), tmat) #one feature with two discrete levels "L" and "R"
obsv = list(c("L","L","R","R","L")) #emissions for the one feature for the 5 nodes in order 1:5
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(hmmA,obsv,ft_sq,kn_st)
```

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

Description

Tree is a complex graphical model where we can have multiple parents and multiple children for a node. Hence the order in which the tree 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 tree, if known. Default is 100

Value

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

See Also

`forward`

Examples

```
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 tree
hmmA = initHMM(c("A","B"),list(c("L","R")), tmat) #one feature with two discrete levels "L" and "R"
ft_sq = fwd_seq_gen(hmmA)
```

initHMM *Initializing treeHMM with given parameters*

Description

Initializing treeHMM with given parameters

Usage

```
initHMM(States, Symbols, treemat, startProbs = NULL, transProbs = NULL,
  emissionProbs = NULL)
```

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 treeHMM
Symbols	List containing (M * 1) vectors for discrete values of emissions(where "M" is the possible number of emissions) for each feature variable
treemat	Adjacent Symmetry Matrix that describes the topology of the tree
startProbs	(N * 1) vector containing starting probabilities for the states, where "N" is the possible number of states(Optional). Default is equally probable states
transProbs	(N * N) matrix containing transition probabilities for the states, where "N" is the possible number of states(Optional)
emissionProbs	List of (N * M) matrices containing emission probabilities for the states, for each feature variable(optional). Default is equally probable emissions

Value

List describing the parameters of treeHMM(pi, alpha, beta)

Examples

```
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 tree
states = c("P","N") #"P" represent cases(or positive) and "N" represent controls(or negative)
hmmA = initHMM(states,list(c("L","R")), tmat) #one feature with two discrete levels "L" and "R"
hmmB = initHMM(states, list(c("X","Y")),tmat, c(0.5,0.5), matrix(c(0.7,0.3,0.3,0.7),2,2))
```

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

Description

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

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

```
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 tree
hmmA = initHMM(c("P","N"),list(c("L","R")), tmat) #one feature with two discrete levels "L" and "R"
Transprob = noisy_or(hmmA,c("P","N"),"P") #for transition from P & N simultaneously to P
```

Index