Package ‘discretecdAlgorithm’

March 12, 2020

Type Package
Title Coordinate-Descent Algorithm for Learning Sparse Discrete Bayesian Networks
Version 0.0.7
Description Structure learning of Bayesian network using coordinate-descent algorithm. This algorithm is designed for discrete network assuming a multinomial data set, and we use a multi-logit model to do the regression. The algorithm is described in Gu, Fu and Zhou (2016) <arXiv:1403.2310>.
License GPL (>= 2)
LazyData TRUE
Depends R (>= 3.2.3)
Imports Rcpp (>= 0.11.0), sparsebnUtils (>= 0.0.4), igraph
Suggests testthat,
LinkingTo Rcpp, RcppEigen
RoxygenNote 7.0.2
SystemRequirements C++11
NeedsCompilation yes
Author Jiaying Gu [aut, cre]
Maintainer Jiaying Gu <gujy.lola@gmail.com>
Repository CRAN
Date/Publication 2020-03-12 06:40:02 UTC

R topics documented:

  cd.run ................................................................. 2
  coef_gen ............................................................ 4
  data_gen ............................................................. 5
  generate_discrete_data ............................................. 5
  max_lambda ........................................................... 7

Index 8
cd.run

Description

Structure learning of discrete Bayesian network

Usage

cd.run(
  indata,
  weights = NULL,
  lambdas = NULL,
  lambdas.length = 30,
  whitelist = NULL,
  blacklist = NULL,
  error.tol = 1e-04,
  convLb = 0.01,
  weight.scale = 1,
  upperbound = 100,
  alpha = 3,
  permute = FALSE,
  adaptive = FALSE
)

Arguments

indata A sparsebnData object.
weights Weight matrix. Weight can be the $l_2$ norm of a consistent estimate of $\beta_{j,i}$. See paper Gu et al. (2016) chapter 3.3 for more details. A weight matrix that is set improperly may cause convergence issues and lead to a suboptimal solution.
lambdas Numeric vector containing a grid of lambda values (i.e. regularization parameters) to use in the solution path. If missing, a default grid of values will be used based on a decreasing log-scale. To generate a sequence of lambdas see generate.lambdas. For discrete network, the paper provided a way to calculate a maximum lambda that penalizes all parameters to zero, Gu et al. (2016) chapter 3.4. See function max_lambda for details.
lambdas.length Integer number of values to include in the solution path.
whitelist A two-column matrix of edges that are guaranteed to be in each estimate (a "white list"). Each row in this matrix corresponds to an edge that is to be whitelisted. These edges can be specified by node name (as a character matrix), or by index (as a numeric matrix).
blacklist A two-column matrix of edges that are guaranteed to be absent from each estimate (a "black list"). See argument "whitelist" above for more details.
error.tol Error tolerance for the algorithm, used to test for convergence.
\textbf{cd.run}  

- \texttt{convLb}: Small positive number used in Hessian approximation.
- \texttt{weight.scale}: A positive number to scale weight matrix.
- \texttt{upperbound}: A large positive value used to truncate the adaptive weights. A -1 value indicates that there is no truncation.
- \texttt{alpha}: Threshold parameter used to terminate the algorithm whenever the number of edges in the current DAG estimate is $> \alpha \times \text{ncol(data)}$.
- \texttt{permute}: A bool parameter, default value is FALSE. If TRUE, will randomize order of going through blocks.
- \texttt{adaptive}: A bool parameter, default value is FALSE. If FALSE, a regular lasso algorithm will be run. If TRUE, an adaptive lasso algorithm will be run.

\section*{Details}
Estimate structure of a discrete Bayesian network from observational/interventional data using the CD algorithm described in Gu et al. (2016).

Instead of producing a single estimate, this algorithm computes a solution path of estimates based on the values supplied to \texttt{lambdas} or \texttt{lambdas.length}. This package do not provide a model selection method in this version, users can choose their own model selection criterion. In later version of this package we will provide an empirical model selection method.

This package can handle interventional data by input a list of intervention. See example for more detail.

\section*{Value}
A \texttt{sparsebnPath} object. The CD Algorithm will be stopped if the number of edges exceeds 3 times number of variables.

\section*{Examples}

```r
## Not run:
### Generate some random data
dat <- matrix(rbinom(200, size = 3, prob = 0.4), nrow = 20)
# for observational data
dat_obs <- sparsebnUtils::sparsebnData(dat, type = "discrete")
# for interventional data
data_size <- nrow(dat)
ivn <- lapply(1:data_size, function(x){return(as.integer(x/10))})
# if there is no intervention for an observation, use 0.
# cd algorithm can handle multiple interventions for a single observation.
dat_int <- sparsebnUtils::sparsebnData(dat, ivn = ivn, type = "discrete")

# Run with default settings for observational data
cd.run(indata = dat_obs)
# Run with default settings for interventional data
cd.run(indata = dat_int)
# Run adaptive algorithm for observational data
cd.run(indata = dat_obs, adaptive = TRUE)
```
### Optional: Adjust settings

```r
debugger::n_node <- ncol(dat)
```

# Run algorithm with a given weight
# Careful with this option.
```r
weights <- matrix(1, nrow = n_node, ncol = n_node)
```

# Run with adjusted settings
```r
cd.run(indata = dat_obs, weights = weights, lambdas.length = 10)
```

# Note: Normally, users do not need to change default settings.

```r
## End(Not run)
```

---

**coef_gen**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>coef_gen</td>
<td>coef_gen</td>
</tr>
</tbody>
</table>

**Description**

coefficient generating function

**Usage**

```r
ccoef_gen(edge_list, n_levels = NULL, FUN = NULL, flip = TRUE)
```

**Arguments**

- `edge_list`: a `edgeList` object.
- `n_levels`: a list of number of levels for each node.
- `FUN`: a probability distribution to generate coefficients
- `flip`: a bool parameter. If true, will randomly flip the sign of coefficients.

**Value**

A list of coefficient matrix
**data_gen**

---

**Description**

A function that generates a discrete data set.

**Usage**

```r
data_gen(graph, n, ivn = NULL, n_levels = NULL, params = NULL, FUN = NULL, flip = TRUE)
```

**Arguments**

- `graph`: a `edgeList` object.
- `n`: size of the data set, a scalar.
- `ivn`: a list of intervention for each data point.
- `n_levels`: a list of number of levels for each node, default is binary data set.
- `params`: coefficient list (optional).
- `FUN`: a function to generate magnitude of influence (optional).
- `flip`: a bool parameter. If true, when generating coefficients, will randomly flip the sign of coefficients.

**Value**

data matrix

---

**generate_discrete_data**

---

**Description**

data generating function
generate_discrete_data

Usage

generate_discrete_data(
  graph,
  params,
  n,
  ivn = NULL,
  ivn.rand = TRUE,
  n_levels = NULL
)

Arguments

graph           A edgelist object.
params          Coefficient list.
n               Size of the data set, a scalar
ivn             List of interventions.
ivn.rand        If TRUE, random values will be drawn uniformly for each intervention. Otherwise, these values need to supplied manually in ivn.
n_levels        A vector of number of levels for each node. Default is binary data.

Value

data matrix

Examples

### generate observational data
gr <- sparsebnUtils::random.graph(5, 5) # use sparsebnUtils package to generate a random graph
names(gr) = c("V1", "V2", "V3", "V4", "V5")
nlevels <- c(3, 5, 2, 2, 3)
gr.params <- coef_gen(edge_list = gr, n_levels = nlevels)
data.obs <- discretecdAlgorithm::generate_discrete_data(graph = gr,
  n = 100,
  n_levels = nlevels,
  params = gr.params)

### generate experimental data
ivn <- as.list(c(rep("V1", 50), rep("V2", 50))) # 50 interventions on V1, 50 interventions on V2
data.ivn <- discretecdAlgorithm::generate_discrete_data(graph = gr,
  n = 100,
  n_levels = nlevels,
  params = gr.params,
  ivn = ivn)

### Use pre-specified values for interventions
### In this toy example, we assume that all intervened nodes were fixed to
### the value 1, although this can be any number of course.
ivn.vals <- lapply(ivn, function(x) sapply(x, function(x) 1)) # replace all entries with a 1
max_lambda

```
data.ivn <- discretecdAlgorithm::generate_discrete_data(graph = gr, 
            n = 100, 
            n_levels = nlevels, 
            params = gr.params, 
            ivn = ivn.vals, 
            ivn.rand = FALSE)
```

---

**Description**

A method to calculate the value of maximum lambda along a solution path. See paper Gu et al. (2016) chapter 3.4 for more detail.

**Usage**

```
max_lambda(indata, weights = NULL, weight.scale = 1, upperbound = 100)
```

**Arguments**

- `indata`: A sparsebnData object
- `weights`: Weight matrix
- `weight.scale`: A positive number to scale weight matrix.
- `upperbound`: A large positive value used to truncate the adaptive weights. A -1 value indicates that there is no truncation.

**Value**

The maximum lambda along the solution path.

**Examples**

```r
## Not run:
### Generate some random data
dat <- matrix(rbinom(200, size = 3, prob = 0.4), nrow = 20)
# for observational data
dat <- sparsebnUtils::sparsebnData(dat, type = "discrete")

# generate the maximum lambda
max_lambda(indata = dat)
## End(Not run)
```
Index

cd.run, 2
coef_gen, 4

data_gen, 5
edgeList, 4–6

generate.lambdas, 2
generate_discrete_data, 5
max_lambda, 2, 7
sparsebnPath, 3