Package 'ccdrAlgorithm'

April 12, 2022

Title CCDr Algorithm for Learning Sparse Gaussian Bayesian Networks

Version 0.0.6

Date 2022-04-10

Maintainer Bryon Aragam <sparsebn@gmail.com>

Description Implementation of the CCDr (Concave penalized Coordinate Descent with reparametrization) structure learning algorithm as described in Aragam and Zhou (2015) <https://www.jmlr.org/papers/v16/aragam15a.html>. This is a fast, score-based method for learning Bayesian networks that uses sparse regularization and block-cyclic coordinate descent.

Depends R (>= 3.2.3)

Imports sparsebnUtils (>= 0.0.5), Rcpp (>= 0.11.0), stats, utils

LinkingTo Rcpp

Suggests testthat, graph, igraph, Matrix

URL https://github.com/itsrainingdata/ccdrAlgorithm

BugReports https://github.com/itsrainingdata/ccdrAlgorithm/issues

License GPL (>= 2)

RoxygenNote 7.1.1

NeedsCompilation yes

Author Bryon Aragam [aut, cre],
Dacheng Zhang [aut]

Repository CRAN

Date/Publication 2022-04-12 02:52:30 UTC

R topics documented:

ccdr.run *Main CCDr Algorithm*

Description

Estimate a Bayesian network (directed acyclic graph) from observational data using the CCDr algorithm as described in Aragam and Zhou (2015).

Usage

```
ccdr.run(
  data,
  lambdas = NULL,
  lambdas.length = NULL,
  whitelist = NULL,
  blacklist = NULL,
  gamma = 2,
  error.tol = 1e-04,
  max.iters = NULL,
  alpha = 10,
  betas,
  sigmas = NULL,
  verbose = FALSE
)
```

Arguments

data	Data as sparsebnData object. Must be numeric and contain no missing values.
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 (see also generate.lambdas).
lambdas.length	Integer number of values to include in the solution path. If lambdas has also been specified, this value will be ignored. Note also that the final solution path may contain fewer estimates (see alpha).
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.
gamma	Value of concavity parameter. If gamma > 0, then the MCP will be used with gamma as the concavity parameter. If gamma < 0, then the L1 penalty will be used and this value is otherwise ignored.
error.tol	Error tolerance for the algorithm, used to test for convergence.
max.iters	Maximum number of iterations for each internal sweep.

alpha	Threshold parameter used to terminate the algorithm whenever the number of edges in the current DAG estimate is > alpha * ncol(data).
betas	Initial guess for the algorithm. Represents the weighted adjacency matrix of a DAG where the algorithm will begin searching for an optimal structure.
sigmas	Numeric vector of known values of conditional variances for each node in the network. If this is set by the user, these parameters will not be computed and the input will be used as the "true" values of the variances in the algorithm. Note that setting this to be all ones (i.e. sigmas[j] = 1 for all j) is equivalent to using the least-squares loss.
verbose	TRUE / FALSE whether or not to print out progress and summary reports.

Details

Instead of producing a single estimate, this algorithm computes a solution path of estimates based on the values supplied to lambdas or lambdas.length. The CCDr algorithm approximates the solution to a nonconvex optimization problem using coordinate descent. Instead of AIC or BIC, CCDr uses continuous regularization based on concave penalties such as the minimax concave penalty (MCP).

This implementation includes two options for the penalty: (1) MCP, and (2) L1 (or Lasso). This option is controlled by the gamma argument.

Value

A sparsebnPath object.

Examples

```
### Generate some random data
dat <- matrix(rnorm(1000), nrow = 20)
dat <- sparsebnUtils::sparsebnData(dat, type = "continuous")

# Run with default settings
ccdr.run(data = dat, lambdas.length = 20)

### Optional: Adjust settings
pp <- ncol(dat$data)

# Initialize algorithm with a random initial value
init.betas <- matrix(0, nrow = pp, ncol = pp)
init.betas[1,2] <- init.betas[1,3] <- init.betas[4,2] <- 1

# Run with adjusted settings
ccdr.run(data = dat, betas = init.betas, lambdas.length = 20, alpha = 10, verbose = TRUE)
```

ccdrAlgorithm *ccdrAlgorithm: CCDr Algorithm for Learning Sparse Gaussian Bayesian Networks*

Description

Implements the CCDr structure learning algorithm as described in Aragam and Zhou (2015).

Details

The CCDr algorithm uses sparse regularization (L1 or MCP) to produce a solution path of DAG estimates along a pre-determined grid of hyperparameters. This package implements a single function, `ccdr.run` that runs the main algorithm, and uses `sparsebnUtils` for the underlying data structures and methods.

---

generate_mvn_data *Generate data from a DAG*

Description

Given a Gaussian DAG, generate data from the underlying distribution. Equivalently, generate data from a multivariate normal distribution given one of its SEM. Can generate both observational and intervention data.

Usage

```
generate_mvn_data(graph, params, n = 1, ivn = NULL, ivn.rand = TRUE)
```

Arguments

graph	DAG in `edgeList` format.
params	Vector of parameters. Last p elements correspond to variances (p = number of nodes in `graph`), initial elements correspond to edge weights.
n	Number of samples to draw.
ivn	List of interventions (see `sparsebnData`). Must be a `list` with exactly `n` components.
ivn.rand	If `TRUE`, random N(0,1) values will be drawn for each intervention. Otherwise, these values need to supplied manually in `ivn`.

Details

If `ivn = NULL`, then `n` observational samples are drawn. For each component of `ivn` that is not `NULL`, interventional samples will be drawn with the values of each node specified in the component.

Examples

```
### Generate observational data
gr <- sparsebnUtils::random.graph(5, 5) # use sparsebnUtils package to generate a random graph
gr.params <- runif(10) # there are 5 coefficients + 5 variances
data.obs <- ccdrAlgorithm::generate_mvn_data(graph = gr,
                                             n = 100,
                                             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 <- ccdrAlgorithm::generate_mvn_data(graph = gr,
                                             n = 100,
                                             params = gr.params,
                                             ivn = ivn)

### Use pre-specified values for interventions
###  In this toy example, we assume that all intervened nodes were fixed to
###  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
data.ivn <- ccdrAlgorithm::generate_mvn_data(graph = gr,
                                             n = 100,
                                             params = gr.params,
                                             ivn = ivn.vals,
                                             ivn.rand = FALSE)

### If ivn.rand = FALSE, you must specify values
###  The code below will fail because ivn does not contain any values
### (compare to ivn.vals above).
## Not run:
data.ivn <- ccdrAlgorithm::generate_mvn_data(graph = gr,
                                             n = 100,
                                             params = gr.params,
                                             ivn = ivn,
                                             ivn.rand = FALSE)

## End(Not run)
```

Index