Package 'spacejam'

July 2, 2014

Type Package

Title Sparse conditional graph estimation with joint additive models.

Version 1.1

Date 2012-06-07

Author Arend Voorman

Maintainer Arend Voorman <voorma@uw.edu>

Depends igraph (>= 0.6), splines, Matrix

Description This package provides an extension of conditional
independence (CIG) and directed acyclic graph (DAG) estimation
to the case where conditional relationships are (non-linear) additive models.

License GPL (>= 2)

NeedsCompilation yes

Repository CRAN

Date/Publication 2013-04-16 19:38:01

R topics documented:

spacejam-package *Fit sparse conditional independence graphs with joint additive models*

Description

This package is called 'spacejam', which can be thought to stand for SParse Conditional Estimation with Joint Additive Models. The two main functions of this package, `SJ` and `SJ.dag`, estimate conditional independence graphs and DAGs using flexible node-wise regressions, employing a standardized group lasso to encourage sparsity. Details of the method are given in Voorman, Shojaie and Witten (2013). Graph Estimation with Joint Additive Models.

Details

Package:	spacejam
Type:	Package
Version:	1.1
Date:	2013-04-08
License:	GPL >= 2

The package includes the following functions:

`SJ` and `SJ.dag`:	Estimate the conditional independence graph, or directed acyclic graph, based on observed data
`rdag`:	construct a random directed acyclic graph
`moralize`:	Determine the moral graph, which is the conditional independence graph associated with the directed
`generate.dag.data`:	Generate data from a bayesian network
`plot.SJ`:	Plot SJ and SJ.dag objects.

Author(s)

Arend Voorman

Maintainer: Arend Voorman <voorma@uw.edu>

References

Voorman, Shojaie and Witten (2013). Graph Estimation with Joint Additive Models. Submitted to Biometrika. available on ArXiv or from authors upon request

See Also

`spacejam` `SJ` `generate.dag.data`

Examples

```
#########Create graph and distribution used in Figure 2 of Voorman, Shojaie and Witten (2013):
```

```
p <- 100 #variables
n <- 50 #observations

#Generate Graph
set.seed(20)
g <- rdag(p,80)
mylayout <- layout.fruchterman.reingold(g)

par(mfrow=c(1,2))
plot(g, layout = mylayout, edge.color = "gray50",
        vertex.color = "red", vertex.size = 3, vertex.label = NA,
        edge.arrow.size = 0.4)
plot(moralize(g), layout = mylayout, edge.color = "gray50",
        vertex.color = "red", vertex.size = 3, vertex.label = NA,
        edge.arrow.size = 0.4)

#create a distribution on the DAG using cubic polynomials with random normal coefficients
#with standard deviations of 1, 0.5 and 0.5, (i.e. giving more weight to linear association than quadratic or cubi
data <- generate.dag.data(g,n,basesd=c(1,0.5,0.5))
X <- data$X

#Fit conditional independence graph at one lambda
fit1 <- SJ(X, lambda = 0.6)

#Fit conditional independence graph at 10 (hopefully reasonable) lambdas:
fit2 <- SJ(X, length = 10)

#Fit conditional independence graph using quadratic basis functions:
fit3 <- SJ(X, bfun = function(x){cbind(x,x^2)}, length = 10)

#Fit the DAG using default causal ordering 1:p, and at 10 lambdas
fit4 <- SJ.dag(X, length = 10)
fit4

#plot the DAGs, and the true graph
par(mfrow=c(1,3))
plot(g, layout=mylayout, edge.color = "gray50", vertex.color = "red", vertex.size = 3, vertex.label = NA, edge.ar
plot(fit4, layout = mylayout, which= 4, main= paste0("lambda = ",round(fit4$lambda[4],2) ))
plot(fit4, layout = mylayout, main = "min BIC")

###For additional replications using the same DAG distribution use e.g.
data <- generate.dag.data(g,n,funclist = data$funclist)


#### Screen out edges whose corresponding nodes have low spearman correlation:
# useful for approximating spacejam in high dimesnions
S <- cor(data$X, method = "spearman")
G.max <- S > 0.1
mean(G.max) #~75% of edges removed

system.time({fit.screen <- SJ(X, G.max = G.max,length=20)})
system.time({fit.noscreen <- SJ(X,length = 20)})
```

```
plot(fit.screen, layout=mylayout)
plot(fit.noscreen, layout=mylayout)
```

`generate.dag.data` *Generate nonlinear data from DAGs*

Description

These funtions create distributions on directed acyclic graphs. `rdag` generates a random DAG with a given number of edges by selecting 'nedges' at random from (p choose 2) possible edges, `moralize` gets the moral graph from a DAG, and `generate.dag.data` generates non-linear data by assigning each edge a cubic polynomial basis with random coefficients.

Usage

```
rdag(p, nedges)
moralize(g)
generate.dag.data(g, n, basesd = 1, basemean = 0, bfuns = function(x){cbind(x, x^2, x^3)},
funclist = NULL, usenorm = T)
```

Arguments

`g`	a directed graph, as an 'igraph' object
`p`	number of vertices
`n`	number of observations
`nedges`	number of edges
`bfuns`	the basis functions for the structural equations. Note that when the basis functions generated, they are centered and scaled to have variance 1, so similar coefficients correspond to similar amounts of variance explained. Ignored if 'funclist' is supplied.
`basesd`	standard deviation of the random coefficients assigned to the basis functions. Ignored if 'funclist' is supplied.
`basemean`	means of the (random) coefficients assigned to the basis functions. Ignored if 'funclist' is supplied.
`funclist`	p by p list of functions determining the structural equations. funclist[[i]][[j]](x) is the effect of feature j on feature i. If this is omitted, the functions are generated at random from the bases supplied in bfuns.
`usenorm`	logical. whether to use normal or uniform errors

Details

Multivariate distributions with complicated conditional dependence structures corresponding to a particular graph are difficult to construct in general. However, constructing complicated distributions from a DAG is straighforward. These functions are meant to facilitate construction of complicated distributions on a DAG, and obtain the coresponding conditional independence structure.

generate.dag.data generates Normal(basemean,basesd) coefficients for the basis functions coresponding to each edge. This gives a function f_ij(x_j), which is standardized to have variance 1. Then data is generated for a feature conditioned on its parents by

x_i = sum_(j in parents(i)) f_ij + noise

where 'noise' is by default N(0,1). These data are returned, along with the generated functions which can be used in subsequent simulations.

Value

'rdag' returns an 'igraph' object.

'generate.dat.data' returns a list with two elements: an n by p matrix 'X' and a p by p list 'funclist', where funclist[[i]][[j]] is the effect of feature j on feature i.

'moralize' returns an undirected igraph object.

Author(s)

Arend Voorman

References

Voorman, Shojaie and Witten (2013). Graph Estimation with Joint Additive Models. Submitted to Biometrika. available on ArXiv or from authors upon request

See Also

igraph spacejam SJ plot.SJ

Examples

```
#########Create graph and distribution used in Figure 2 of Voorman, Shojaie and Witten (2013):
p <- 100 #variables
n <- 50 #observations

#Generate Graph
set.seed(20)
g <- rdag(p,80)
mylayout <- layout.fruchterman.reingold(g)

par(mfrow=c(1,2))
plot(g, layout = mylayout, edge.color = "gray50",
        vertex.color = "red", vertex.size = 3, vertex.label = NA,
        edge.arrow.size = 0.4)
plot(moralize(g), layout = mylayout, edge.color = "gray50",
        vertex.color = "red", vertex.size = 3, vertex.label = NA,
```

```
        edge.arrow.size = 0.4)

#create a distribution on the DAG using cubic polynomials with random normal coefficients
#with standard deviations of 1, 0.5 and 0.5, (i.e. giving more weight to linear association than quadratic or cubi
data <- generate.dag.data(g,n,basesd=c(1,0.5,0.5))
X <- data$X

#Fit conditional independence graph at one lambda
fit1 <- SJ(X, lambda = 0.6)

###For additional replications using the same DAG distribution use e.g.
data <- generate.dag.data(g,n,funclist = data$funclist)
```

`plot.SJ` *plot an object of class* `SJ` *or* `SJ.dag`

Description

This function plots an SJ or SJ.dag

Usage

```
## S3 method for class 'SJ'
plot(x, which = NULL, layout = NULL, ...)
```

Arguments

`x`	an object of class SJ or SJ.dag
`which`	The index of which to plot. If not specified, the penalty with the smallest BIC is used.
`layout`	the layout of the graph to use. If not specified, `layout.fruchterman.reingold` is used.
`...`	additional parameters to be passed to `plot.igraph`.

Details

This function plots a graph in SJ or SJ.dags 'graph' field, using some sane degaults for vertex size and color.

Value

returns the layout used, invisibly.

Author(s)

Arend Voorman

References

Voorman, Shojaie and Witten (2013). Graph Estimation with Joint Additive Models. Submitted to Biometrika. available on ArXiv or from authors upon request

See Also

plot.igraph spacejam SJ generate.dag.data

Examples

```
p <- 100 #variables
n <- 50 #observations

#Generate Data
set.seed(20)
g <- rdag(p,80)
data <- generate.dag.data(g,n,basesd=c(1,0.5,0.5))
X <- data$X

#Fit conditional independence graph for sequence of 10 lambdas
fit1 <- SJ(X, length = 10)

par(mfrow=c(1,2))
layout <- plot(fit1, main = "min BIC")
plot(fit1, which=5, layout = layout, main = paste0("lambda = ",round(fit1$lambda[5],3)))
```

SJ *Estimate a graph with Spacejam*

Description

These functions estimate graphs using flexible node-wise regressions, employing a group-lasso penalty to encourage sparsity. Details are given in Voorman, Shojaie and Witten (2013).

Usage

```
SJ(X, bfun = bs, lambda=NULL, length =NULL, verbose = FALSE, b0 = NULL, maxit = 100,
tol = .Machine$double.eps^0.25,G.max = NULL)

SJ.dag(X, bfun = bs, lambda=NULL, length =NULL, ord = 1:p, verbose = FALSE, b0 = NULL,
maxit = 100, tol = .Machine$double.eps^0.25)
```

Arguments

X	an n by p matrix of observations
bfun	a function to generate bases for each of the p features (e.g. 'bs' or `function(x){cbind(x,x^2)}`). The default is bs, which generates cubic polynomials (i.e. spacejam in 3 dimensions)

lambda	penalty terms. These should be between 0 and k, where k is the number of basis functions. If none are specified, the 'length' argument specifies the number of penalty terms to choose automatically (default is 10).
length	number of penalty terms. This is ignored if lambda is specified.
verbose	logical. whether to print progress indicator.
b0	a p*k by p matrix of coefficients to be used as initial values. Typically obtained from the 'betas' value of an SJ object.
maxit	maximum number of iterations
tol	Tolerance for convergence. This is the maximum change in coefficients in subsequent iterations.
ord	for SJ.dag: The causal ordering of the variables, according to the columns of X. If none is given, assumed to be 1:ncol(X) i.e. the first column of X is first in the causal ordering, and the final column is last in the causal ordering.
G.max	An adjacency matrix of dimension ncol(X) by ncol(X) specifying all potential edges to be considered. This is a Useful for screening rules, where many edges are not considered, or incorporating known structures of the graph.

Details

This implements the method described in Voorman, Shojaie and Witten 'Graph estimation with joint additive models', which regresses each feature on the others using generalized additive models, subject to sparsity inducing penalties.

The function is designed to estimate the graph for a sequence of tuning parameters, using warm starts to improve speed. In addition, an 'active set' approach is used as described in Friedman et al (2010).

Value

an object of class 'SJ' or 'SJ.dag'.

Among some internal variables, these objects include the elements

graphs	a list of the estimates, each of class igraph
lambda	a vector of the corresponding tuning parameters
bic	a vector of estimated BIC criteria.
dfs	a p x length matrix of the estimated degrees of freedom for each feature and tuning parameter.
rss	a p x length matrix of the residual sum of squares for each feature and tuning parameter.

Author(s)

Arend Voorman

References

Voorman, Shojaie and Witten (2013). Graph Estimation with Joint Additive Models. Submitted to Biometrika. available on ArXiv or from authors upon request

Jerome Friedman, Trevor Hastie, Robert Tibshirani (2010). Regularization Paths for Generalized Linear Models via Coordinate Descent. Journal of Statistical Software, 33(1), 1-22. URL http://www.jstatsoft.org/v33/i01/.

See Also

spacejam generate.dag.data plot.SJ

Examples

```
#########Create graph and distribution used in Figure 2 of Voorman, Shojaie and Witten (2013):
p <- 100 #variables
n <- 50 #observations

#Generate Graph
set.seed(20)
g <- rdag(p,80)
mylayout <- layout.fruchterman.reingold(g)

par(mfrow=c(1,2))
plot(g, layout = mylayout, edge.color = "gray50",
        vertex.color = "red", vertex.size = 3, vertex.label = NA,
        edge.arrow.size = 0.4)
plot(moralize(g), layout = mylayout, edge.color = "gray50",
        vertex.color = "red", vertex.size = 3, vertex.label = NA,
        edge.arrow.size = 0.4)

#create a distribution on the DAG using cubic polynomials with random normal coefficients
#with standard deviations of 1, 0.5 and 0.5, (i.e. giving more weight to linear association than quadratic or cubi
data <- generate.dag.data(g,n,basesd=c(1,0.5,0.5))
X <- data$X

#Fit conditional independence graph at one lambda , using the default basis functions (cubic polynomials).
fit1 <- SJ(X, lambda = 0.6)

#Fit conditional independence graph at 10 (hopefully reasonable) lambdas:
fit2 <- SJ(X, length = 10)

#Fit conditional independence graph using quadratic basis functions:
fit3 <- SJ(X, bfun = function(x){cbind(x,x^2)}, length = 10)

#Fit the DAG using default causal ordering 1:p, and at 10 lambdas
fit4 <- SJ.dag(X, length = 10)
fit4

#plot the DAGs, and the true graph
par(mfrow=c(1,3))
plot(g, layout=mylayout, edge.color = "gray50", vertex.color = "red", vertex.size = 3, vertex.label = NA, edge.ar
```

```
plot(fit4, layout = mylayout, which= 4, main= paste0("lambda = ",round(fit4$lambda[4],2) ))
plot(fit4, layout = mylayout, main = "min BIC")

###For additional replications using the same DAG distribution use e.g.
data <- generate.dag.data(g,n,funclist = data$funclist)


#### Screen out edges whose corresponding nodes have low spearman correlation:
# useful for approximating spacejam in high dimesnions
S <- cor(data$X, method = "spearman")
G.max <- S > 0.1
mean(G.max) #~75% of edges removed

system.time({fit.screen <- SJ(X, G.max = G.max,length=20)})
system.time({fit.noscreen <- SJ(X,length = 20)})

plot(fit.screen, layout=mylayout)
plot(fit.noscreen, layout=mylayout)
```

Index