Package 'GCnetinf'

May 3, 2015

Type Package

Title Granger causality inference

Version 1.0

Date 2015-05-03

Author Miguel Lopes

Maintainer Miguel Lopes <miguelaglopes@gmail.com>

Description Takes as input a multiple time series, returns dynamic causality scores between variables.

Imports tseries, ppcor, randomForest, lars

License GPL-3

R topics documented:

`GCnetinf-package` *GCnetinf*

Description

This package computes dynamic causality scores between variables from a multiple time series matrix. Consists of two main functions, described below. The first function is "gcausal" and estimates linear Granger causality (GC) scores. Bivariate and conditional (1 order) GC tests are available, returning a matrix of z-scores, where the element [i,j] is the score from variable i to j. The conditional GC score from a cause to effect is the minimum 1 order conditional GC score obtained, for all individual conditioning variables. An heuristic is applied in order to speed up this search, as described in Lopes 2015. Options for lag selection and integrated variables are also available. The second function is "netinf1l" and implements dynamic versions (1-lag) of state of the art network inference algorithms: bivariate mutual information, aracne, mrmr, cmim, mimr, random forests, and lasso/least angle regression (assessed in Lopes 2015).

Details

Package:	GCnetinf
Type:	Package
Version:	1.0
Date:	2015-05-03
License:	GPL3

See the entries ?gcausal and ?netinf11 on how to use this package.

Author(s)

Miguel Lopes

Maintainer: Miguel Lopes <miguelaglopes@gmail.com>

References

Lopes 2015 (PhD Thesis)

`gcausal` *Granger causality (GC) tests for multiple time series.*

Description

This function estimates bivariate and conditional (1 order) Granger causality between each pair of variables in the dataset. Returns a p times p matrix of GC z-scores, obtained with an F-test on the residuals of restricted and unrestricted models. The conditional GC scores are the minimum of first order conditional GC scores (a score designated by GC3 herein). In order to improve its speed in the large variable case, an approximation is implemented based on a search heuristic described in Lopes 2015. Its accuracy and speed can be controlled with a user given parameter (see below). It is possible to filter variable pairs which are identified as having common causes (siblings). This identification is described in Lopes 2015 (which is referred to as co-regulation identification in the context of gene regulations). In this case, scores between siblings are assigned NA values.

GC tests may consider multiple lags and correction for integrated variables (Toda-Yamamoto test). The default option is to consider a single first lag, but this may be modified (see below).

The methods implemented are described in detail in Lopes 2015 (PhD thesis).

Usage

```
gcausal(datamatrix,
type = "conditional",
lagmethod = "first",
maxnumlags = 1,
maxlag = 1,
crit = "aicc",
ty.test = FALSE,
int.maxorder = 2,
stat.method = "KPSS",
stat.cutoff = 0.05,
sibling.filter = FALSE,
sf.mincor = 0.7,
```

```
sf.maxlag = 2,
rank.method = "dynamic",
search.speed = 3,
sf.matrix,
rank.matrix)
```

Arguments

datamatrix	A numeric matrix of dimension n times p (samples are rows, variables are columns). No NA or Inf values allowed.
type	Character string, either "bivariate" or "conditional" (default).
lagmethod	Character string, either "first (default) or "fsel". The lag(s) of the target are always the first (its number may be estimated by AIC). Then, an equal number of predictor lags are considered. However, these may be selected in a forward selection procedure, instead of being the first. For instance, the lags returned by "first" may be 1,2,3, and the lags returned by "fsel" 2,1,4.
maxnumlags	Integer (default 1). This parameter defines the maximum number of lags (of a variable) to be included in the GC model.
maxlag	Integer (default 1). This parameter defines the maximum lag to be included in the GC model (it may be different than maxnumlags when lagmethod="fsel").
crit	Character string, either "aicc" (default), "aic" or "bic". This parameter defines the criterion to assess linear models.
ty.test	Logical, TRUE or FALSE (default FALSE). If TRUE, the Toda-Yamamoto modified GC test is used (deals with integrated variables).
int.maxorder	Integer (default 2). This parameter defines the maximum order of integration in the integration tests.
stat.method	Character string, either "KPSS" (default) or "adf". This parameter defines the method to test null hypothesis of stationarity.
stat.cutoff	Numeric (default 0.05). P-value level to reject stationarity.
sibling.filter	Logical, TRUE or FALSE (default). If TRUE scores between siblings (variables with common causes) are filtered (the respective scores are NA). Sibling identification is as described in Lopes 2015.
sf.mincor	Numeric (default 0.7). This parameter defines the minimum linear correlation for sibling identification.
sf.maxlag	Integer (default 2). This parameter defines the maximum considered lag in sibling identification.
rank.method	Character string, either "static" or "dynamic" (default). This parameter defines the method to compute the ranking in GC3.
search.speed	Integer (default 3). This parameter controls the speed of the GC3 search ("t" in Lopes 2015). The lower the faster, and less precise. Maximum is p (equivalent to the full search).
sf.matrix	(optional) p times p sibling matrix (if already computed). Non zero elements indicate siblings.
rank.matrix	(optional) Ranking matrix for GC3 (if already computed)

Value

A p times p matrix of GC z-scores. Score [i,j] is from element i to element j.

Author(s)

Miguel Lopes

References

Lopes 2015 (PhD Thesis)

Examples

```
#datamatrix=mat.or.vec(50,20)+rnorm(50*20,0,1)
#gcscores=gcausal(datamatrix, type="bivariate")
```

`netinf1l` *1-lag dynamic network inference algorithms for time series.*

Description

This function implements 1-lag dynamic network inference algorithms for time series. For each target, predictors are lagged (1 lag), and scored according to the selected model. Returns a p \times p matrix of scores. Score [i,j] is from variable i to j.

Implemented methods are bivariate mutual information, aracne, mrmr, cmim, mimr, random forests, and lasso/least angle regression. The last two call the packages randomForest and lars. As described in PhD thesis Lopes 2015.

Usage

```
netinf1l(datamatrix,
Methods,
mi.cutoff = 0.05,
rf.importance = "IncNodePurity",
rf.mtry = round(sqrt(ncol(exprdata))),
rf.ntrees = 1000,
lars.type = "lasso",
lars.use.gram = TRUE,
lars.mode = "fraction",
lars.path = seq(0.1, 1, 0.01))
```

Arguments

`datamatrix`	A numeric matrix of dimension n times p (samples are rows, variables are columns). No NA or Inf values allowed.
`Methods`	Character string, either "mi" (mutual information) "aracne", "mrmr", "cmim", "mimr", "rf" (random forests), or "lars" (least angle regression). This parameter selects the used inference method.
`mi.cutoff`	Numeric (default 0.05). The mutual information is estimated as a function of the linear correlation (Gaussian assumption). This value is the significancy cut-off for the statistical test on liner correlations.
`rf.importance`	Character string, one of "%IncMSE" and "IncNodePurity". Measure of variable importance using randomForest.

rf.mtry	Parameter "mtry" in the function randomForest::randomForest (default sqrt(number of variables)).
rf.ntrees	Parameter "ntrees" in the function randomForest::randomForest (default 1000).
lars.type	Parameter "type" in the function lars::lars (default "lasso")
lars.use.gram	Parameter "use.gram" in the function lars::lars (default TRUE).
lars.mode	Parameter "mode" in the function lars::lars (default "fraction").
lars.path	Parameter "path" in the function lars::lars (default seq(0.1, 1, 0.01)).

Value

* A p times p matrix of network scores. Score [i,j] is from element i to element j.

Author(s)

Miguel Lopes

References

Lopes 2015 (PhD Thesis)

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (datamatrix, Methods, mi.cutoff = 0.05, rf.importance = "IncNodePurity",
    rf.mtry = round(sqrt(ncol(exprdata))), rf.ntrees = 1000,
    lars.type = "lasso", lars.use.gram = TRUE, lars.mode = "fraction",
    lars.path = seq(0.1, 1, 0.01))
{
    if (any(is.na(datamatrix))) {
        stop("NA values found")
    }
    if (any(abs(datamatrix) == Inf)) {
        stop("Inf values found")
    }
    exprdata = t(datamatrix)
    predictions = list()
    if (any(c("mi", "aracne", "clr") %in% Methods)) {
        mi.mat = gci.cor2mi(gci.cor.mat(exprdata))
    }
    for (m in 1:length(Methods)) {
        method = Methods[m]
        if (method == "mi") {
            predictions[[m]] = mi.mat
        }
        if (method == "aracne") {
            predictions[[m]] = gci.aracne(mi.mat)
        }
        if (method == "clr") {
            predictions[[m]] = gci.clr(mi.mat)
        }
        if (method == "mrmr") {
```

```
            predictions[[m]] = gci.mrmr(exprdata)
        }
        if (method == "cmim") {
            predictions[[m]] = gci.cmim(exprdata, mi.cutoff)
        }
        if (method == "mimr") {
            predictions[[m]] = gci.mimr(exprdata, mi.cutoff)
        }
        if (method == "rf") {
            predictions[[m]] = gci.rf(exprdata, rf.importance,
                rf.mtry, rf.ntrees)
        }
        if (method == "lars") {
            predictions[[m]] = gci.lars(exprdata, lars.type,
                lars.use.gram, lars.mode, lars.path)
        }
    }
    names(predictions) = Methods
    REM = which(unlist(lapply(predictions, is.null)))
    if (length(REM) > 0) {
        predictions = predictions[-REM]
    }
    for (m in 1:length(predictions)) {
        diag(predictions[[m]]) = min(predictions[[m]])
        predictions[[m]] = predictions[[m]] - min(predictions[[m]])
    }
    return(predictions)
}
```

Index