Package ‘SINGLE’

February 19, 2015

Type Package

Title Estimate sparse dynamic graphs using the Smooth Incremental Graphical Lasso Estimation (SINGLE) algorithm

Version 1.3

Date 2013-12-04

Author Ricardo Pio Monti, Christoforos Anagnostopoulos and Giovanni Montana

Maintainer Ricardo Pio Monti <ricardo.monti08@gmail.com>

Imports

Depends flsa, igraph, dse, Matrix, R (>= 2.10.1)

Description This package provides an implementation of the Smooth Incremental Graphical Lasso Estimation (SINGLE) which can be used to estimate dynamic networks from non-stationary time series data.

License GPL-2

NeedsCompilation no

Repository CRAN

Date/Publication 2014-05-28 18:30:13

R topics documented:

SINGLE-package .................................................. 2
choose_h ......................................................... 3
generate_random_data .......................................... 4
get_kern_cov ..................................................... 5
plotSINGLE ....................................................... 6
precision_recall ............................................... 7
SINGLE ............................................................ 8

Index 11
Estimate dynamic networks using the Smooth Incremental Graphical Lasso Estimation (SINGLE) Algorithm

Description

This package provides an implementation of the Smooth Incremental Graphical Lasso Estimation (SINGLE) Algorithm which can be used to accurately estimate dynamic networks from non-stationary time series data.

Details

Package: SINGLE
Type: Package
Version: 1.0
Date: 2013-10-13
License: GPL-2

The SINGLE algorithm is indeed

Author(s)

Ricardo Pio Monti, Christoforos Anagnostopoulos and Giovanni Montana
Maintainer: Ricardo Pio Monti <ricardo.monti08@gmail.com>

References


See Also

SINGLE, plotSINGLE, generate_random_data, precision_recall

Examples

set.seed(1)
X = generate_random_data(ROI=5, length_=50, seg=3, sparsity=0.15, str=0.6)
## Not run:
S = SINGLE(data=X$data, radius=5, l1=0.75, l2=0.3, k=3)
result = precision_recall(true_cov=X$true_cov, estimated_cov=S$P_)
plot(result$F1, type='l', ylim=c(0,1), ylab=''
main='F1 Score', xlab='Time') # plot F1 score over time

## End(Not run)
choose_h

Estimate kernel width for Gaussian kernel and Sliding window

Description
Estimate the optimal width of kernel/sliding window using look-ahead log-likelihood

Usage
choose_h(data, sample_size, kernel = "gaussian", h)

Arguments
- **data**: Data used to estimate optimal width. Each row should be an observation.
- **sample_size**: Number of samples to use when computing the look-ahead log-likelihood. This is done in order to reduce the computational cost. In order to use all samples simply set sample_size = nrow(data).
- **kernel**: Type of kernel to use. Options are "gaussian" for Gaussian kernel and "window" for sliding window.
- **h**: A vector of potential kernel widths.

Details
We estimate the optimal value of $h$ by looking to maximise the look-ahead log-likelihood. This is defined for the $i$th observation as follows:

$$ L_{i-1}(i; h) = -\frac{1}{2} \log \det (S_{i-1}) - \frac{1}{2} (X_i - \mu_{i-1})^T S_{i-1}^{-1} (X_i - \mu_{i-1}) $$

We note that both $\mu_{i-1}$ and $S_{i-1}$ are estimated with the $i$th observation removed. It follows that calculating $L_{i-1}(h)$ for all observations is computationally expensive, thus in order to save computational effort we sample a subset of observations $R \subseteq \{2, \ldots, T\}$. The optimal value of $h$ can then be estimated by choosing $h$ to maximise the following score function:

$$ CV(h) = \sum_{i \in R} L_{i-1}(i; h), $$

where in order to reduce the variability of estimated look-ahead log-likelihoods we use the same sample $R$ to calculate each $CV(h)$.

Value
Optimal value of $h$ is returned

Author(s)
Ricardo Pio Monti
generate_random_data

Generate a random dynamic graph using Vector Autoregressive (VAR) processes

Description

This function generates random data with a dynamic correlation structure. Given the number of changepoints, the correlation structure within each segment of data (i.e., between changepoints) is randomly simulated using Erdos-Renyi random graphs.

Usage

generate_random_data(ROI, length_, mode='ER', seg, sparsity, str = 0.6)

Arguments

ROI Number of regions of interest (i.e., number of nodes in the simulated graph)
length_ The length of each segment of data between changepoints
mode The type of random network to generate. Options are 'ER' for Erdos-Renyi or 'BA' for Barabasi-Albert scale-free networks.
seg Number of data segments (each with its own randomly sampled correlation structure)
sparsity Sparsity parameter when simulating Erdos-Renyi random graphs. This is the probability of an edge occurring between each pair of nodes. Should be between 0 and 1.
str The strength of the correlation between nodes. This should be between -1 and 1. We note that in the case of Erdos-Renyi random graphs all edges between nodes will have a strength of str. However, in the case of Barabasi-Albert scale-free networks the strength of each edge will follow a uniform distribution on \([\frac{-str}{2}, \frac{str}{2}] \cup [\frac{str}{2}, str]\).
get_kern_cov

Value

data A matrix containing the simulated data
true_cov An array where true_cov[,i] contains the adjacency matrix corresponding
to the true correlation structure at the ith observation

Author(s)

Ricardo Pio Monti

See Also

SINGLE, plotSINGLE, precision_recall, choose_h, get_kern_cov

Examples

set.seed(1)
X = generate_random_data(ROI=10, length_=100, seg=3, sparsity=0.1, str=0.6)
## Not run:
S = SINGLE(data=X$data, h=50, l1=0.7, l2=0.5)
precision_recall(true_cov=X$true_cov, estimated_cov=S$P)
## End(Not run)

get_kern_cov

Estimate time varying sample covariance matrices using kernels

Description

Estimate sample covariance matrices using Guassian kernel or sliding window.

Usage

get_kern_cov(data, h, kernel = "gaussian")

Arguments

data Matrix containing data from which to estimate time varying sample covariance
matrices. Each row should contain an observation
h Kernel width
kernel Choice of kernel. Options are "gaussian" for Gaussian kernel and "window" for Sliding window.

Details

The ith sample covariance matrix $S_i$ is estimated as:

$$S_i = \frac{\sum_{j=1}^{T} K_h(i, j) \cdot (X_j - \mu_j)^T (X_j - \mu_j)}{\sum_{j=1}^{T} K_h(i, j)}$$
Value
A list where the ith element is the sample covariance at the ith observation

Author(s)
Ricardo Pio Monti

See Also
SINGLE, generate_random_data, choose_h

Examples
```r
set.seed(1)
X = generate_random_data(ROI=5, length_=50, seg=3, sparsity=0.15, str=0.6)
C = get_kern_cov(data=X$data, h=50, kernel="gaussian")
## Not run:
S = SINGLE(C=C, l1=0.7, l2=0.5)
plotSINGLE(S, index = c(1,2,3,4,5), x.axis=seq(1,150),
          col.names=seq(1,5), n.row=2, fix.axis=TRUE)
## End(Not run)
```

plotSINGLE

Plot pairwise partial correlations as estimated by the SINGLE algorithm

Description
Plot pairwise partial correlations as estimated by the SINGLE algorithm

Usage
```r
plotSINGLE(object, index, x.axis, col.names, n.row, fix.axis = FALSE)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An object of class SINGLE.</td>
</tr>
<tr>
<td>index</td>
<td>Subset of indices for which we wish to plot partial correlations. If is.na(index) then all partial correlations are plotted, however this may be a bit crowded.</td>
</tr>
<tr>
<td>x.axis</td>
<td>X axis for partial correlation plots</td>
</tr>
<tr>
<td>col.names</td>
<td>Names for columns. This must match the index.</td>
</tr>
<tr>
<td>n.row</td>
<td>Will plot pairwise partial correlations on a grid, use this to determine number of rows on the grid.</td>
</tr>
<tr>
<td>fix.axis</td>
<td>Should axis on all plots be the same, this allows for easier comparison of partial correlations.</td>
</tr>
</tbody>
</table>
precision_recall

Details

We note that given an estimate of a precision matrix, \( \hat{\Theta} \), the partial correlation between the \( i \) and \( j \) nodes, \( p_{i,j} \), will be

\[
p_{i,j} = -\frac{\Theta_{ij}}{\sqrt{\Theta_{ii}\Theta_{jj}}}
\]

Author(s)

Ricardo Pio Monti

See Also

SINGLE, generate_random_data, precision_recall

Examples

```r
set.seed(1)
X = generate_random_data(ROI=5, length_=50, seg=3, sparsity=0.1, str=0.6)
## Not run:
S = SINGLE(data=X$data, h=0.7, l1=0.5)
plotSINGLE(S, index = c(1,2,3,4,5), x.axis=seq(1,150),
            col.names=seq(1,5), n.row=2, fix.axis=TRUE)
## End(Not run)
```

description

This function calculates the Precision, Recall and F scores by comparing the true graphical structure to the estimated graphical structure.

Usage

```r
precision_recall(true_cov, estimated_cov)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true_cov</td>
<td>An array where <code>true_cov[,i]</code> is the adjacency matrix of the true graphical</td>
</tr>
<tr>
<td></td>
<td>structure at the <code>i</code>th observation</td>
</tr>
<tr>
<td>estimated_cov</td>
<td>An array where <code>estimated_cov[,i]</code> is the adjacency matrix of the estimated</td>
</tr>
<tr>
<td></td>
<td>graphical structure at the <code>i</code>th observation</td>
</tr>
</tbody>
</table>
In our case, Precision is defined as the fraction of reported edges which are true edges and Recall is the fraction of edges which are correctly reported. We calculate the F scores as $F_i = \frac{P_i \cdot R_i}{(P_i + R_i)}$ where $P_i$ and $R_i$ are the precision and recall at the $i$th observation respectively.

**Value**

- **pres**: A vector where the $i$th entry is the precision of the estimated graph at the $i$th observation
- **recall**: A vector where the $i$th entry is the recall of the estimated graph at the $i$th observation
- **F1**: A vector where the $i$th entry is the F score of the estimate graph at the $i$th observation

**Author(s)**

Ricardo Pio Monti

**See Also**

*SINGLE*, *plotSINGLE*, *generate_random_data*

**Examples**

```r
set.seed(1)
X = generate_random_data(ROI=5, length_=50, seg=3, sparsity=0.1, str=0.6)
## Not run:
S = SINGLE(data=X$data, h=50, l1=0.7, l2=0.5)
result = precision_recall(true_cov=X$true_cov, estimated_cov=S$p_)
plot(result$F1, type='l', ylim=c(0,1), ylab='',
     main='F1 Score', xlab='Time') # plot F1 score over time
## End(Not run)
```

---

**SINGLE**

*Smooth Incremental Graphical Lasso Estimation (SINGLE)*

**Description**

Estimate sparse dynamic graphs using the Smooth Incremental Graphical Lasso Estimation (SINGLE) algorithm

**Usage**

`SINGLE(data, C, h, l1, l2, kernel="gaussian", tol = 0.01, verbose=FALSE)`
Arguments

data
- Matrix of observations where each row is a vector of observations.
- (Optional) Array of estimated sample covariance matrices. If provided then these are used directly in the SINGLE algorithm. Format should be such that \( c_{[i, 1]} \) is the estimated sample covariance matrix at the ith observation. Alternatively, if c is not provided then sample covariance matrices will be estimated from data.

h
- (Optional) Width of the Gaussian kernel. h can either be numeric or a vector. If h is numeric then this value will be used to estimate sample covariance matrices. If a vector is provided then the optimal h will be estimated using the choose_h function. If h is missing, the optimal value will be estimated using choose_h function for some values based on the dimensions of data. Finally, if c is provided, this will be ignored.

l1
- (Optional) Value for the \( \lambda_1 \) penalty. l1 can either be numeric or a vector. If l1 is numeric then this value will be used directly. If a vector is provided then the optimal l1 will be estimated by minising AIC. If l1 is missing, the optimal value will be estimated over a range of values.

lR
- (Optional) Value for the \( \lambda_2 \) penalty. lR can either be numeric or a vector. If lR is numeric then this value will be used directly. If a vector is provided then the optimal lR will be estimated by minising AIC. If lR is missing, the optimal value will be estimated over a range of values.

kernel
- (Optional) Choice of kernel. Can be either "gaussian" for a Gaussian kernel or "window" for a sliding window. If omitted a Gaussian kernel will be used.

tol
- Threshold for convergence.

verbose
- Flag for printing out information as iterations proceed.

Details

The SINGLE algorithm minimises the following objective function:

\[
\text{f}(\{\Theta\}) = \sum_{i=1}^{T} \left( \log det \hat{\Theta}_i + \text{trace}(S_i \Theta_i) \right) + \lambda_1 \sum_{i=1}^{T} ||\Theta_i||_1 + \lambda_2 \sum_{i=2}^{T} ||\Theta_i - \Theta_{i-1}||_1
\]

where \( \{\Theta\} \) is the set of all precision matrices indexed over time.

Value

\( P \)
- List where the ith element is the ith estimated precision matrix

\( P_\cdot \)
- An array where \( P_\cdot_{[i, 1]} \) is the estimated precision matrix for the ith observation

\( C_\cdot \)
- List where the ith element is the ith estimated covariance matrix

l1
- \( \lambda_1 \) parameter used for final fit. If l1 not given this is the estimated \( \lambda_1 \) value.

l2
- \( \lambda_2 \) parameter used for final fit. If l2 not given this is the estimated \( \lambda_2 \) value.

h
- Kernel width parameter used for final fit. If h not given this is the estimated kernel width.

AIC
- AIC for final model based on parameters \( \lambda_1, \lambda_2 \) and h.
Author(s)
Ricardo Pio Monti

References

See Also
plotSINGLE, generate_random_data, precision_recall, choose_h

Examples
set.seed(1)
X = generate_random_data(ROI=5, length_=50, seg=3, sparsity=0.1, str=-0.6)
## Not run:
S = SINGLE(data=X$data, verbose=TRUE)
plotSINGLE(S, index=c(1,2,3,4,5), fix.axis=T)
## End(Not run)
Index

*Topic **SINGLE**
  plotSINGLE, 6
  SINGLE, 8
*Topic **kernel**
  choose_h, 3
  get_kern_cov, 5
choose_h, 3, 5, 6, 10

generate_random_data, 2, 4, 4, 6–8, 10
get_kern_cov, 4, 5, 5

plotSINGLE, 2, 5, 6, 8, 10
precision_recall, 2, 5, 7, 7, 10

SINGLE, 2, 4–8, 8
SINGLE-package, 2