Package 'TGS'

May 7, 2026

Version 1.0.1

Title Rapid Reconstruction of Time-Varying Gene Regulatory Networks

Description Rapid advancements in high-throughput gene sequencing technologies have resulted in genome-scale time-series datasets. Uncovering the underlying temporal sequence of gene regulatory events in the form of time-varying gene regulatory networks demands accurate and computationally efficient algorithms. Such an algorithm is 'TGS'. It is proposed in Saptarshi Pyne, Alok Ranjan Kumar, and Ashish Anand. Rapid reconstruction of time-varying gene regulatory networks. IEEE/ACM Transactions on Computational Biology and Bioinformatics, 17(1):278{291, Jan-Feb 2020. The TGS algorithm is shown to consume only 29 minutes for a microarray dataset with 4028 genes. This package provides an implementation of the TGS algorithm and its variants.

License CC BY-NC-SA 4.0

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0

biocViews NetworkInference, GraphAndNetwork, Network, GeneExpression, Microarray, SystemsBiology, Software

URL https://www.biorxiv.org/content/early/2018/06/14/272484, https://github.com/sap01/TGS

BugReports https://github.com/sap01/TGS/issues

Imports rjson, bnstruct, ggm, foreach, doParallel, minet (>= 3.38.0)

Suggests R.rsp, testthat (>= 2.1.0), knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author Saptarshi Pyne [aut, cre] (ORCID: <https://orcid.org/0000-0001-9710-6749>),
Manan Gupta [aut],
Alok Kumar [aut],
Ashish Anand [aut] (ORCID: <https://orcid.org/0000-0002-0024-3358>)

Maintainer Saptarshi Pyne <saptarshipyne01@gmail.com>

Repository CRAN

Date/Publication 2020-05-07 13:00:21 UTC

Contents

`TGS-package` *TGS: A package for Rapid Reconstruction of Time-Varying Gene Regulatory Networks*

Description

The `TGS` package provides an implementation of the TGS algorithm and its variants. This algorithm reconstructs time-varying gene regulatory networks from time-series gene expression datasets. For algorithmic details, please see: Saptarshi Pyne, Alok Ranjan Kumar, and Ashish Anand. Rapid reconstruction of time-varying gene regulatory networks. IEEE/ACM Transactions on Computational Biology and Bioinformatics, 17(1):278–291, Jan–Feb 2020.

The `LearnTgs` function

Please call `TGS::LearnTgs()` to invoke the TGS algorithm. See the manual for the required parameters.

`LearnTgs` *Implement the TGS Algorithm*

Description

The TGS algorithm takes a time-series gene expression dataset as input. It analyses the data and reconstructs the underlying temporal sequence of gene regulatory events. The reconstructed output is given in the form of time-varying gene regulatory networks (GRNs). The TGS algorithm is extremely time-efficient and hence suitable for processing large datasets with hundreds to thousands of genes. More details about the algorithm can be found at Saptarshi Pyne, Alok Ranjan Kumar, and Ashish Anand. Rapid reconstruction of time-varying gene regulatory networks. IEEE/ACM Transactions on Computational Biology and Bioinformatics, 17(1):278–291, Jan–Feb 2020.

Usage

```
LearnTgs(
  isfile = 0,
  json.file = "",
  input.dirname = "",
  input.data.filename = "",
  num.timepts = 2,
  true.net.filename = "",
  input.wt.data.filename = "",
  is.discrete = TRUE,
  num.discr.levels = 2,
  discr.algo = "",
  mi.estimator = "mi.pca.cmi",
  apply.aracne = FALSE,
  clr.algo = "CLR",
  max.fanin = 14,
  allow.self.loop = TRUE,
  scoring.func = "BIC",
  output.dirname = ""
)
```

Arguments

`isfile` Numeric. 1 or 0. 1 if input arguments are given in a json file. Otherwise, 0.

`json.file` Character string. Absolute path to the JSON file if `isfile = 1`.

`input.dirname` Character string. Absolute path to the directory where input files are kept. By default, the current working directory.

`input.data.filename`

Character string. Name of the file containing the input data. It can either be a '.tsv' file or an '.RData' file.

- If it is a '.tsv' file, the first column should have the time point IDs. The only exception is the (1, 1)-th cell. This cell should be reserved for the column header. The column header can be anything, such as - 'Time'. The first row, excluding the (1, 1)-th cell, must have the gene names. The rest of the cells should contain the corresponding value. For example, the cell with row ID 't1' and column ID 'G2' would represent the expression value of gene 'G2' at time point 't1'.
- If it is an '.RData' file, the underlying object should be a matrix named 'input.data'. In 'input.data', the row names must represent the time point IDs. On the other hand, the column names must represent the gene names. Therefore, the (i, j)-th cell of the matrix contains the expression value of the j-th gene at the i-th time point.

For both '.tsv' and '.RData' input, multiple rows with the same time point ID represent multiple replicates at the same time point. In other words, these rows belong to the same time point in different time series. The time points belonging to the same time series must be together and in ascending order. An exemplary

dataset with three genes {G1, G2, G3}, two time points {t1, t2} and two time series is shown below.

Time	G1	G2	G3
t1	0.8272480342	0.7257430901	0.3894130418
t2	0.6542518342	0.6470658823	0.5088904888
t1	0.3519554463	0.3551279726	0.3207993604
t2	0.4871730974	0.3706990326	0.447523615

`num.timepts` Numeric. Positive integer greater than 1. Number of distinct time points.

`true.net.filename`

Character string. Name of the file containing the true network. In case it is non-empty, the name should refer to an '.RData' file. The '.RData' file must have an object named 'true.net.adj.matrix'. The object can either be a matrix or a list.

- If the object is a matrix, then it represents the true summary GRN. The row names and column names should be the gene names. Each cell can contain a value of 1 or 0. If the (i, j)-th cell contains 1, then there exists an edge from the i-th gene to the j-th gene. Otherwise, the edge does not exist. An example with three genes {G1, G2, G3} is given below.

	G1	G2	G3
G1	0	1	0
G2	0	0	0
G3	1	0	0

- If the object is a list, then it represents the true time-varying GRNs. The length of the list must be equal to the number of time intervals, which is `(num.timepts - 1)`. Each element in the list should be a matrix. The p-th matrix represents the true GRN corresponding to the p-th time interval. The row names and column names should be the gene names. Each cell can contain a value of 1 or 0. If the (i, j)-th cell contains 1, then there exists an edge from the i-th gene to the j-th gene. Otherwise, the edge does not exist.

`input.wt.data.filename`

Character string. Name of the file containing the Wild Type expressions of the genes. If non-empty, then must be a '.tsv' file. The first row should contain the names of the genes. Only exception is the (1, 1)-th cell which should be empty. The second row should have the wild type expressions. Therefore, the (2, j)-th cell must contain the wild type expression of the j-th gene. Again the only exception is the (2, 1)-th cell which should be empty. An example with three genes {G1, G2, G3} is given below.

<empty>	G1	G2	G3
<empty>	0.5298713	0.5174261	0.8181522

`is.discrete` Logical. TRUE or FALSE. TRUE if the input data is discrete. Otherwise, FALSE.

`num.discr.levels`
: Numeric. Positive integer greater than 1. Number of discrete levels that each gene has (if the input data is discrete) or each gene should have (if the input data needs to be discretised).

`discr.algo`
: Character string. Name of the discretisation algorithm to be used when the input data needs to be discretised. The available algorithms are – 'discretizeData.2L.Tesla' and 'discretizeData.2L.wt.l'. If you choose algorithm 'discretizeData.2L.wt.l', please provide the wild type data using argument `input.wt.data.filename`.

`mi.estimator`
: Character string. Name of the algorithm for estimating mutual informations. There is only one algorithm available at this moment. It is 'mi.pca.cmi'.

`apply.aracne`
: Logical. TRUE or FALSE. TRUE if you wish to apply ARACNE for refining the mutual information matrix. Otherwise, FALSE.

`clr.algo`
: Character string. Name of the context likelihood relatedness (CLR) algorithm to use. The available algorithms are – 'CLR', 'CLR2', 'CLR2.1', 'CLR3' and 'spearman'.

`max.fanin`
: Numeric. Positive integer. Maximum number of regulators each gene can have.

`allow.self.loop`
: Logical. TRUE or FALSE. TRUE if you wish to allow self loops. Otherwise, FALSE.

`scoring.func`
: Character string. Name of the scoring function to use. At this moment, the only available option is 'BIC'.

`output.dirname`
: Character string. File path to a directory where output files are to be saved. There are three options. *Option 1:* It can be the absolute path to an existing directory. *Option 2:* It can also be the absolute path to a non-existing directory. In this case, the directory will be created. *Option 3 (default):* If provided an empty string, then it will be the current working directory.

Details

The function does not return any values. Instead, it outputs a set of files and saves them under the directory specified by `output.dirname`. The output files are described in Section 'Value'.

Value

input.data.discr.RData Discretised version of the input data. This file is created only if the input data is not discretised as specified by input argument 'is.discrete'.

mut.info.matrix.RData Mutual information matrix of the given genes. This RData file contains a matrix named 'mut.info.matrix'. The (i, j)-th cell of the matrix represents the mutual information between the i-th and j-th genes. This is a symmetric matrix.

mi.net.adj.matrix.wt.RData Weighted Mutual information network of the given genes. This RData file contains a matrix named 'mi.net.adj.matrix.wt'. The (i, j)-th cell of the matrix represents the weight of the edge from the i-th gene to the j-th gene. The edge weight is a non-negative real number.

mi.net.adj.matrix.RData Unweighted Mutual information network of the given genes. This RData file contains a matrix named 'mi.net.adj.matrix'. Each cell of the matrix contains a value of 1 or 0. If the (i, j)-th cell contains 1, then there exists an edge from the i-th gene to the j-th gene. Otherwise, the edge does not exist.

unrolled.DBN.adj.matrix.list.RData Reconstructed time-varying GRNs. This RData file contains a list named 'unrolled.DBN.adj.matrix.list'. The length of the list is equal to the total number of time intervals, which is (num.timepts - 1). Each element in the list is a network adjacency matrix. The p-th element in the list represents the adjacency matrix of the GRN corresponding to the p-th time interval. In this adjacency matrix, each cell contains a value of 1 or 0. If the (i, j)-th cell contains 1, then there exists a directed edge from the i-th gene to the j-th gene. Otherwise, the edge does not exist.

di.net.adj.matrix.RData Rolled GRN. This RData file contains a matrix named 'di.net.adj.matrix'. Each cell in the matrix contains a value of 1 or 0. If the (i, j)-th cell contains 1, then there exists an edge from the i-th gene to the j-th gene. Otherwise, the edge does not exist.

net.sif Rolled GRN in the SIF format compatible with Cytoscape.

Result.RData Correctness metrics. This file is created only if true network is given through input argument 'true.net.filename'. Inside this RData file, there is a matrix named 'Result'. The columns represent the correctness metrics, such as - TP (number of true positive predictions) and FP (number of false positive predictions). The rows depend upon the nature of the true network. If the true network is time-varying GRNs, then the number of rows is equal to the number of time intervals. In that case, the p-th row contains the correctness metrics of the reconstructed GRN corresponding to the p-th time interval. On the other hand, if the true network is a summary GRN, then there exists only one row. This row represents the correctness metrics of the rolled GRN.

output.txt Console output.

sessionInfo.txt R session information.

Examples

```
## Not run:
  TGS::LearnTgs(
  isfile = 0,
  json.file = '',
  input.dirname = 'C:/GitHub/TGS/inst/extdata',
  input.data.filename = 'InSilicoSize10-Yeast1-trajectories.tsv',
  num.timepts = 21,
  true.net.filename = 'DREAM3GoldStandard_InSilicoSize10_Yeast1_TrueNet.RData',
  input.wt.data.filename = 'InSilicoSize10-Yeast1-null-mutants.tsv',
  is.discrete = FALSE,
  num.discr.levels = 2,
  discr.algo = 'discretizeData.2L.wt.l',
  mi.estimator = 'mi.pca.cmi',
  apply.aracne = FALSE,
  clr.algo = 'CLR',
  max.fanin = 14,
  allow.self.loop = FALSE,
  scoring.func = 'BIC',
  output.dirname = 'C:/GitHub/TGS/inst/extdata/Output_Ds10n')

  TGS::LearnTgs(
  isfile = 0,
  json.file = '',
  input.dirname = 'C:/GitHub/TGS/inst/extdata',
```

```
  input.data.filename = 'edi-data-10n.tsv',
  num.timepts = 21,
  true.net.filename = 'edi.net.10.adj.mx.RData',
  input.wt.data.filename = '',
  is.discrete = FALSE,
  num.discr.levels = 2,
  discr.algo = 'discretizeData.2L.Tesla',
  mi.estimator = 'mi.pca.cmi',
  apply.aracne = FALSE,
  clr.algo = 'CLR',
  max.fanin = 14,
  allow.self.loop = TRUE,
  scoring.func = 'BIC',
  output.dirname = 'C:/GitHub/TGS/inst/extdata/Output_Ed10n')

## End(Not run)
```

Index