Package ‘ts2net’

October 14, 2022

Title From Time Series to Networks

Version 0.1.0

Description Transforming one or multiple time series into networks. This package is useful for complex systems modeling, time series data mining, or time series analysis using networks.


License MIT + file LICENSE

URL https://github.com/lnferreira/ts2net

BugReports https://github.com/lnferreira/ts2net/issues

Encoding UTF-8

RoxygenNote 7.1.2

Depends R (>= 4.1.0), igraph (>= 1.2.11), parallel, compiler

Imports dtw (>= 1.22.3), scales (>= 1.1.1), minerva (>= 1.5.10),
infotheo (>= 1.2.0), mmpp (>= 0.6), dbscan (>= 1.1.10), zoo (>= 1.8.9), nonlinearTseries (>= 0.2.11), stats, utils

Suggests covr, testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation no

Author Leonardo N. Ferreira [aut, cre]

Maintainer Leonardo N. Ferreira <ferreira@leonardonascimento.com>

Repository CRAN

Date/Publication 2022-06-09 07:40:02 UTC
dataset_sincos_generate

Sin-Cos data set generator. This function generates a set of sine and cosine time series. This function is used as example of the package application.

Description

Sin-Cos data set generator. This function generates a set of sine and cosine time series. This function is used as example of the package application.
**Usage**

```r
dataset_sincos_generate(
  num_sin_series = 25,
  num_cos_series = 25,
  x_max = 8 * pi,
  ts_length = 100,
  jitter_amount = 0.1,
  return_x_values = FALSE
)
```

**Arguments**

- `num_sin_series`: Integer. Number of sine time series
- `num_cos_series`: Integer. Number of cosine time series
- `x_max`: Float. Max x value in sin(x) or cor(x).
- `jitter_amount`: Float. The total amount of jitter added to each time series.
- `return_x_values`: Boolean. If positive, returns a list of data frames with x and y values.

**Value**

A list with all time series. First the `num_sin_series` sine time series followed by the `num_cos_series` cosine time series.

---

**dist_file_parts_merge  Merge parts of distances stored in files.**

**Description**

The functions `tsdist_dir_parallel` and `tsdist_parts_parallel` calculate part of the distance matrix `D`. The results of the multiple calls of these functions are normally stored in RDS or csv files. This function merges these files and construct a distance matrix `D`.

**Usage**

```r
dist_file_parts_merge(list_files, dir_path, num_elements, file_type = "RDS")
```

**Arguments**

- `list_files`: A list of files with distances.
- `dir_path`: If `list_files` was not passed, than this function uses this parameter to read the files in this directory.
- `num_elements`: The number of time series in the data set. The number of elements defines the number of rows ans columns in the distance matrix `D`. 
**dist_parts_merge**

The extension of the files where the distances are stored. It can be "RDS" (default) or "csv". The RDS files should be data frames composed by three columns i, j, and dist. This format is preferred because it is a compact file. The other option is a "csv" also containing the i, j, and dist columns.

**Value**

Distance matrix D

**dist_matrix_normalize**

Normalize a distance/similarity matrix.

**Description**

Normalize a distance/similarity matrix.

**Usage**

```r
dist_matrix_normalize(D, to = c(0, 1))
```

**Arguments**

- **D** Distance/similarity matrix
- **to** An array of two elements c(min_value, max_value) representing the interval where the elements of dist_matrix will be normalized to.

**Value**

Normalized matrix

**dist_parts_merge**

Merge parts of distances stored in data frames.

**Description**

The functions tsdist_dir_parallel and tsdist_parts_parallel calculate part of the distance matrix D. This function merges these files and construct a distance matrix D.

**Usage**

```r
dist_parts_merge(list_dfs, num_elements)
```

**Arguments**

- **list_dfs** A list of data frames. Each data frame should have three columns i, j, and dist.
- **num_elements** The number of time series in the data set. The number of elements defines the number of rows ans columns in the distance matrix D.
**Value**

Distance matrix D

---

**Description**

Returns the distance value that corresponds to the desired percentile. This function is useful when the user wants to generate networks with different distance functions but with the same link density.

**Usage**

dist_percentile(D, percentile = 0.1, is_D_symetric = TRUE)

**Arguments**

- **D** distance matrix
- **percentile** (Float) The desired percentile of lower distances.
- **is_D_symetric** (Boolean)

**Value**

Distance percentile value.

---

**events_from_ts**

Extract events from a time series.

---

**Description**

This function transforms an time series (array) into a binary time series where 1 means a event and 0 means no event.

**Usage**

events_from_ts(
    ts,
    th,
    method = c("greater_than", "lower_than", "top_percentile", "lower_percentile", "highest", "lowest"),
    return_marked_times = FALSE
)

---
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ts</code></td>
<td>Array. Time series</td>
</tr>
<tr>
<td><code>th</code></td>
<td>A threshold (if <code>method=greater_than</code> or <code>=lower_than</code>), or the percentile (if <code>method=top_percentile</code> or <code>=lower_percentile</code>), or the total number (if <code>method=highest</code> or <code>=lowest</code>).</td>
</tr>
<tr>
<td><code>method</code></td>
<td>String. One of following options: * ‘greater_than’: All values greater or equal to ‘th’. * ‘lower_than’: All values lower or equal to ‘th’. * ‘top_percentile’: Values greater than the ‘th’ percentile. * ‘highest’: The top ‘th’ values. * ‘lowest’: The lower ‘th’ values.</td>
</tr>
<tr>
<td><code>return_marked_times</code></td>
<td>Return the time indices (marked points) where the events occur.</td>
</tr>
</tbody>
</table>

Value

An event (binary, 1: event, 0 otherwise) time series

---

net_enn  

Construct an epsilon-network from a distance matrix.

Description

Construct an epsilon-network from a distance matrix.

Usage

```r
net_enn(  
  D,  
  eps,  
  treat_NA_as = 1,  
  is_dist_symetric = TRUE,  
  weighted = FALSE,  
  invert_dist_as_weight = TRUE,  
  add_col_rownames = TRUE  
)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>D</code></td>
<td>Distance matrix</td>
</tr>
<tr>
<td><code>eps</code></td>
<td>the threshold value to be considered a link. Only values lower or equal to epsilon become 1.</td>
</tr>
<tr>
<td><code>treat_NA_as</code></td>
<td>A numeric value, usually 1, that represent NA values in the distance matrix</td>
</tr>
<tr>
<td><code>is_dist_symetric</code></td>
<td>Boolean, TRUE (default) if dist is symmetric</td>
</tr>
<tr>
<td><code>weighted</code></td>
<td>Boolean, TRUE will create a weighted network</td>
</tr>
</tbody>
</table>
invert_dist_as_weight
Boolean, if weighted == TRUE, then the weights become 1 - distance. This is the default behavior since most network measures interpret higher weights as stronger connection.

add_col_rownames
Boolean. If TRUE (default), it uses the column and row names from dist matrix as node labels.

Value
a igraph network

net_enn_approx

Construct an approximated epsilon neighbor network (faster, but approximated) from a distance matrix. Some actual nearest neighbors may be omitted.

Description
Construct an approximated epsilon neighbor network (faster, but approximated) from a distance matrix. Some actual nearest neighbors may be omitted.

Usage
net_enn_approx(D, eps, ...)

Arguments
D Distance matrix
eps (Integer) k nearest-nearest neighbors where each time series will be connected to
... Other parameters to frNN() function from dbscan package.

Value
Approximated epsilon nearest-neighbor network
net_knn

Construct a knn-network from a distance matrix.

Description
Construct a knn-network from a distance matrix.

Usage
net_knn(D, k, num_cores = 1)

Arguments
D Distance matrix
k (Integer) k nearest-nearest neighbors where each time series will be connected to
num_cores (Integer) Number of cores to use.

Value
k nearest-neighbor network

net_knn_approx

Construct an approximated knn-network (faster, but approximated) from a distance matrix.

Description
Construct an approximated knn-network (faster, but approximated) from a distance matrix.

Usage
net_knn_approx(D, k, ...)

Arguments
D Distance matrix
k (Integer) k nearest-nearest neighbors where each time series will be connected to
... Other parameters to kNN() function from dbscan package.

Value
Approximated k nearest-neighbor network
**net_weighted**

*Creates a weighted network.*

**Description**

A link is created for each pair of nodes, except if the distance is maximum (1). In network science, stronger links are commonly represented by high values. For this reason, the link weights returned are 1 - D.

**Usage**

```r
net_weighted(D, invert_dist_as_weight = TRUE)
```

**Arguments**

- `D` Distance matrix. All values must be between [0,1].
- `invert_dist_as_weight` Boolean, if weighted == TRUE, then the weights become 1 - distance. This is the default behavior since most network measures interpret higher weights as stronger connection.

**Value**

Fully connected network

---

**random_ets**

*Random event time series generator*

**Description**

It generates an event time series with length `ts_length` with `num_events` events considering a uniform probability distribution.

**Usage**

```r
random_ets(ts_length, num_events, return_marked_times = FALSE)
```

**Arguments**

- `ts_length` Time series Length
- `num_events` The number of events
- `return_marked_times` Return the time indices (marked points) where the events occur.

**Value**

An event (binary, 1: event, 0 otherwise) time series
### tsdist_ccf

**Cross-correlation distance**

**Description**

Minimum correlation distance considering a +/- max lag (lag_max)

**Usage**

```r
tsdist_ccf(
  ts1,
  ts2,
  type = c("correlation", "covariance"),
  cor_type = "abs",
  directed = FALSE,
  lag_max = 10,
  return_lag = FALSE
)
```

**Arguments**

- **ts1**: Array. Time series 1
- **ts2**: Array. Time series 2
- **type**: String. "correlation" or "covariance" to be used (type) in the ccf function.
- **cor_type**: String. "abs" (default), "+", or "-". "abs" considers the correlation absolute value. "+" only positive correlations and "-" only negative correlations.
- **directed**: Boolean. If FALSE (default), the lag interval [-lag_max,+lag_max] is considered. Otherwise, [-lag_max,0] is considered.
- **lag_max**: Integer. Default = 10.
- **return_lag**: Also returns the time lag that leads to the shortest distances.

**Value**

Distance

---

### tsdist_cor

**Absolute, positive, or negative correlation distance.**

**Description**

Considering r the person correlation coefficient, this function returns either 1 - abs(r) if cor_type=="abs", 1 - pmax(0, r) if cor_type == "+", or 1 - pmax(0, r * -1) if cor_type == "-". Another possibility is to run a significance test to verify if the r is significant.
tsdist_dtw

Usage

    tsdist_cor(ts1, ts2, cor_type = "abs", sig_test = FALSE, sig_level = 0.01, ...)

Arguments

    ts1        Array. Time series 1
    ts2        Array. Time series 2
    cor_type   String. "abs" (default), "+", or ".-". "abs" considers the correlation absolute
                value. "+" only positive correlations and ".-" only negative correlations.
    sig_test   Run a statistical test. Return 0 if significant or 1 otherwise.
    sig_level  The significance level to test if correlation is significant.
    ...        Additional parameters to cor.test() function.

Value

    Real value [0,1] where 0 means perfect positive (or negative if positive_cor==FALSE) correlation
    and 1 no positive (or negative if positive_cor==FALSE) correlation.

-------------------

tsdist_dtw        Dynamic Time Warping (DTW) distance.
-------------------

Description

    This function is a wrapper for the dtw() function from the dtw package.

Usage

    tsdist_dtw(ts1, ts2, ...)

Arguments

    ts1        Array. Time series 1
    ts2        Array. Time series 2
    ...        Additional parameters for the dtw() function from the dtw package.

Value

    DTW distance
Event synchronization distance test.

Description

Usage
tsdist_es(
  ets1,
  ets2,
  tau_max = +Inf,
  method = c("quiroga", "boers"),
  sig_test = FALSE,
  reps = 100,
  sig_level = 0.01
)

Arguments
ets1 Event time series 1 (one means an event, or zero otherwise)
ets2 Event time series 2 (one means an event, or zero otherwise)
tau_max The maximum tau allowed ()
method "quiroga" (default) for the default co-occurrence count and normalization or "boers" for the co-occurrence count with tau_max and no normalization.
sig_test Run a significance test. Return 0 if significant or 1 otherwise.
reps Number of repetitions to construct the confidence interval
sig_level The significance level to test if correlation is significant.

Details

Value
distance
tsdist_mic

Maximal information coefficient (MIC) distance.

Description

This function transforms the MIC function (from minerva package) into a distance function.

Usage

tsdist_mic(ts1, ts2)

Arguments

ts1  Array. Time series 1
ts2  Array. Time series 2

Value

Distance

---

tsdist_nmi

Normalized mutual information distance

Description

Calculates the normalized mutual information (NMI) and returns it as distance 1 - NMI.

Usage

tsdist_nmi(
  ts1,
  ts2,
  nbins = c("sturges", "freedman-diaconis", "scott"),
  normalization = c("sum", "min", "max", "sqrt"),
  method = "emp"
)

Arguments

ts1  Array. Time series 1
ts2  Array. Time series 2
nbins  The number of bins used for the discretization of both time series. It can be a positive integer or a string with one of the three rules "sturges" (default), "freedman-diaconis", or "scott".
**normalization**

The mutual information (I) normalization method. Options are "sum" (default) $1-(2I/(h1+h2))$, "min" $1-(I/min(h1,h2))$, "max" $1-(I/max(h1,h2))$, and "sqrt" $1-(I/sqrt(h1*h2))$.

**method**

The name of the entropy estimator used in the functions mutinformation() and entropy() from the infotheo package.

---

**Value**

Distance

---

**tsdist_voi**

*Variation of Information distance*

**Description**

The variation of information (VoI) is a distance function based on mutual information.

**Usage**

```r
tsdist_voi(
  ts1,
  ts2,
  nbins = c("sturges", "freedman-diaconis", "scott"),
  method = "emp"
)
```

**Arguments**

- **ts1**
  
  Array. Time series 1

- **ts2**
  
  Array. Time series 2

- **nbins**
  
  The number of bins used for the discretization of both time series. It can be a positive integer or a string with one of the three rules "sturges" (default), "freedman-diaconis", or "scott".

- **method**
  
  The name of the entropy estimator used in the functions mutinformation() and entropy() from the infotheo package.

**Value**

Distance
tsdist_vr

**van Rossum distance**

**Description**

This function compares the times which the events occur e.g., time indices where the time series values are different than zero. Note that the intensity does not matter but if there is an event or not. This function also performs a statistical test using a shuffling approach to test significance. This implementation uses the fmetric function from the mmpp package.

**Usage**

```r
tsdist_vr(ets1, ets2, tau = 1, sig_test = FALSE, reps = 100, sig_level = 0.01)
```

**Arguments**

- **ets1**: Event time series 1 (one means an event, or zero otherwise)
- **ets2**: Event time series 2 (one means an event, or zero otherwise)
- **tau**: Parameter for filtering function (See fmetric function from mmpp package.)
- **sig_test**: Run a statistical test. Return 0 if significant or 1 otherwise.
- **reps**: Number of repetitions to construct the confidence interval
- **sig_level**: The significance level to test if correlation is significant.

**Value**

- **distance**

---

tsnet_rn

**Construct the recurrence network from a time series.**

**Description**

This function constructs the recurrence matrix of the time series using the function ‘rqa()’ from nonlinearTseries package.

**Usage**

```r
tsnnet_rn(x, radius, embedding.dim, time.lag = 1, do.plot = FALSE, ...)
```
Arguments

- **x**: Array. Time series
- **radius**: Maximum distance between two phase-space points to be considered a recurrence.
- **embedding.dim**: Integer denoting the dimension in which we shall embed the time series. If missing, the embedding dimensions is estimated using `estimateEmbeddingDim()` from **nonlinearTseries**. The constructed igraph network has the estimated dimension (and other info) as a parameter. For example: `net$embedding_dim`
- **time.lag**: Integer denoting the number of time steps that will be use to construct the Takens' vectors.
- **do.plot**: Boolean. Show recurrence plot (default = FALSE)
- **...**: Other parameters to 'rqa()' from **nonlinearTseries**

Value

recurrence network

---

**tsnet_vg**

*Construct the visibility graph from a time series*

Description

TODO: weights

Usage

`tsnet_vg(x, method = c("nvg", "hvg"), limit = +Inf, num_cores = 1)`

Arguments

- **x**: Array. Time series
- **method**: String. Construction method: "nvg" (default) for Natural visibility graph, "hvg" horizontal visibility graph.
- **limit**: Positive integer. The maximum temporal distance (indexes) allowed in the visibility. This parameter limits the max visibility.
- **num_cores**: Number of cores (default = 1).

Value

visibility graph
**tssim_event_sync**  
*Event synchronization measure*

**Description**

This function is an adapted version of the coocmetric function from the package mmpp. The differences are the introduction of a tau_max limitation factor and the optional normalization.

**Usage**

```r
tssim_event_sync(
  tts1,
  tts2,
  tau_max = 1,
  normalization = c("both", "min", "none")
)
```

**Arguments**

- `tts1`: Time indices marking events in time series 1
- `tts2`: Time indices marking events in time series 2
- `tau_max`: Max tau to be considered
- `normalization`: Forms of normalization after the co-occurrence count. Possible values "both" (default), "min", and "none". The Default is "both", the original normalization defined by Quiroga et al: \sqrt{N1*N2}. This normalization might be problematic when both time series have very different number of events. Another possibility is to normalize the count by the "min" length between both series. The interpretation now takes into account only the series with less events. For example, considering two series, one with many events and another with just a single event, the results can be 1 (total sync). The option "none" means no normalization and the method returns the total count of synchronized events.

**Details**


**Value**

Synchronization-based similarity
ts_dist

Calculate distances between pairs of time series in a list.

Description

This function calculates the distance between all combinations of time series in the list and returns a distance matrix. This function is usually the first try and might work if the number of time series and their length are not too big.

Usage

```r
ts_dist(
  ts_list,
  dist_func = tsdist_cor,
  is_symetric = TRUE,
  error_value = NaN,
  warn_error = TRUE,
  num_cores = 1,
  ...
)
```

Arguments

- **ts_list** List of time series (arrays).
- **dist_func** Function to be applied to all combinations of time series. This function should have at least two parameters for each time series. Ex: function(ts1, ts2) cor(ts1, ts2)
- **is_symetric** Boolean. If the distance function is symmetric.
- **error_value** The value returned if an error occur when calculating a the distance for a pair of time series.
- **warn_error** Boolean. If TRUE (default), a warning will rise when an error occur during the calculations.
- **num_cores** Numeric. Number of cores
- **...** Additional parameters for measureFunc

Value

A distance or similarity matrix $M$ whose position $M_{ij}$ corresponds to distance or similarity value between time series $i$ and $j$. 
ts_dist_part

*Calculate distances between pairs of time series in part of a list.*

**Description**

This function is particularly useful to run in parallel as jobs in a cluster (HPC). It returns a data frame with elements (i,j) and a distance value calculated for the time series i and j. Not all the elements are calculated but just a part of the total combinations of time series in the list. This function load all the time series in the memory to make the calculations faster. However, if the time series are too long and/or the dataset is huge, it might represent a memory problem. In this case, `dist_dir_parallel()` is more recommended.

**Usage**

```r
ts_dist_part(
  ts_list,  # List of time series.
  num_part,  # Numeric positive between 1 and the total number of parts (num_total_parts).
  num_total_parts,  # This value corresponds to the part (chunk) of the total number of parts to be calculated.
  combinations,  # A list composed by arrays of size 2 indicating the files indices to be compared.
  dist_func = tsdist_cor,  # Function to be applied to all combinations of time series. This function should have at least two parameters for each time series. Ex: function(ts1, ts2) cor(ts1, ts2)
  isSymetric = TRUE,
  error_value = NaN,
  warn_error = TRUE,
  simplify = TRUE,
  num_cores = 1,
  ...)
```

**Arguments**

- **ts_list** List of time series.
- **num_part** Numeric positive between 1 and the total number of parts (num_total_parts). This value corresponds to the part (chunk) of the total number of parts to be calculated.
- **num_total_parts** Numeric positive corresponding the total number of parts.
- **combinations** A list composed by arrays of size 2 indicating the files indices to be compared. If this parameter is passed, then the function does not split all the possibilities and does not use the parameters num_part and num_total_parts. This parameter is useful when the number of combinations is very high and this functions is called several times (high num_total_parts). In this case, instead of calculating all the combinations in each call, the user can calculate it once and pass it via this parameter.
- **dist_func** Function to be applied to all combinations of time series. This function should have at least two parameters for each time series. Ex: function(ts1, ts2) cor(ts1, ts2)
isSymetric  Boolean. If the distance function is symmetric.
error_value  The value returned if an error occur when calculating a the distance for a pair of time series.
warn_error  Boolean. If TRUE (default), a warning will rise when an error occur during the calculations.
simplify  Boolean. If FALSE, returns a list of one (if isSymetric == FALSE) or two elements (if isSymetric == TRUE).
num_cores  Numeric. Number of cores
...  Additional parameters for measureFunc

Value

A data frame with elements (i,j) and a distance value calculated for the time series i and j.

Descriptio

This function works similarly as dist_parts_parallel(). The difference is that it reads the time series from RDS files in a directory. The advantage of this approach is that it does not load all the time series in memory but reads them only when necessary. This means that this function requires much less memory and should be preferred when memory consumption is a concern, e.g., huge data set or very long time series. The disadvantage of this approach is that it requires a high number of file read operations which considerably takes more time during the calculations. IMPORTANT: the file order is very important so it is highly recommended to use numeric names, e.g., 0013.RDS.

Usage

```r
  ts_dist_part_file(
    input_dir, 
    num_part, 
    num_total_parts, 
    combinations, 
    measureFunc = tsdist_cor, 
    isSymetric = TRUE, 
    error_value = NaN, 
    warn_error = TRUE, 
    simplify = TRUE, 
    num_cores = 1, 
    ...
  )
```
ts_to_windows

Arguments

input_dir  Directory path for the directory with time series files (RDS)
num_part  Numeric positive between 1 and the total number of parts (num_total_parts). This value corresponds to the part (chunk) of the total number of parts to be calculated.
num_total_parts  Numeric positive corresponding the total number of parts.
combinations  A list composed by arrays of size 2 indicating the files indices to be compared. If this parameter is passed, then the function does not split all the possibilities and does not use the parameters num_part and num_total_parts.
measureFunc  Function to be applied to all combinations of time series. This function should have at least two parameters for each time series. Ex: function(ts1, ts2) cor(ts1, ts2)
isSymetric  Boolean. If the distance function is symmetric.
error_value  The value returned if an error occur when calculating a the distance for a pair of time series.
warn_error  Boolean. If TRUE (default), a warning will rise when an error occur during the calculations.
simplify  Boolean. If FALSE, returns a list of one (if isSymetric == FALSE) or two elements (if isSymetric == TRUE).
um_cores  Numeric. Number of cores
...  Additional parameters for measureFunc

Value

A data frame with elements (i,j) and a distance value calculated for the time series i and j. Each index corresponds to the order where the files are listed.

Extract time windows from a time series

description

This function is useful when constructing a network from a single time series. The returned list can be directly used to calculate the distance matrix D with ts_dist().

usage

ts_to_windows(x, width, by = 1)

Arguments

x  time series
width  window length
by  Window step. This is the number of values in and out during the window rollover process.
ts_to_windows

Value
  List of windows
Index

dataset_sincos_generate, 2
dist_file_parts_merge, 3
dist_matrix_normalize, 4
dist_parts_merge, 4
dist_percentile, 5
events_from_ts, 5
net_en, 6
net_en_approx, 7
net_knn, 8
net_knn_approx, 8
net_weighted, 9
random_ets, 9
ts_dist, 18
ts_dist_part, 19
ts_dist_part_file, 20
ts_to_windows, 21
tsdist_ccf, 10
tsdist_cor, 10
tsdist_dtw, 11
tsdist_es, 12
tsdist_mic, 13
tsdist_nmi, 13
tsdist_voi, 14
tsdist_vr, 15
tsnet_rn, 15
tsnet_vg, 16
ts_sim_event_sync, 17