SBM-for-correlation-based-networks Documentation

Katharina Baum

Mar 04, 2019

CONTENTS:

1 Indices and tables 1

Python Module Index 9

Index 11

CHAPTER

ONE

INDICES AND TABLES

- genindex
- modindex
- search

calc_edge_prob.**compute_single_edge_prob**(*SBM_in*, *compute_all_spurious=True*, *compute_all_missing=True*, *disconn_nodes=[]*, *miss_edge_list=[]*, *spur_edge_list=[]*, *multiprocess_nodecount=1*, *maxedges_perjob=100000*)

Function to compute the single edge confidence scores (edge probabilities) of a stochastic block model using multiprocessing. We do not consider weighted models here!

Parameters

- SBM_in: an SBM for which to compute the edge probabilities
- compute_all_spurious: Boolean, if edge probabilities for all spurious links should be computed
- compute_all_missing: Boolean, if edge probabilities for all missing links should be computed
- disconn_nodes: list of node indices of disconnected nodes. These are important for computation of missing edge probabilities; the last one will be used to compute all missing edge probabilities with connected nodes, the last and the first for the edge probability for edges connecting disconnected nodes (all the same!). Note that for graphs read in via read_graph() and giving the total number of nodes, the disconnected nodes will be those with largest index.
- miss_edge_list: only needed if compute_spurious==False (if ==True, all possible missing edges are used), list of putatively missing edges for which to determine the missing edge probabilities, has always 2 elements:
  - miss_edge_list[0] is the list of edges between connected nodes,
  - miss_edge_list[1] is the list of edges between disconnected nodes
- spur_edge_list: only needed if compute_missing=False (if == True, all existing edges are determined), list of list with putatively spurious edges for which to determine the spurious edge probabilities as first entry, second entry for list of edges to disconnected nodes
- multiprocess_nodecount: number of processes which are spawned, default=1 (no multiprocessing)
- maxedges_perjob: number of egdes whose probability should be computed per job. If too low, overhead by starting multiple jobs will be high, if too high, runtime can get high. We used 10000 for networks with 8000+ nodes and 2million+ edges.

Returns

A list of three elements:

- spurious edge probabilities
- missing edge probabilities between connected nodes
- missing edge probability between a disconnected and otherwise connected nodes

Of the third, the last element will be the edge probability between two disconnected nodes. Each of the three elements of the list will be a structured, named numpy array for the corresponding edge type in which the source node index is given ('source', int), the target node index ('target', int) and the rounded edge probability ('edgeprob', f4).

Examples

Let sbm_metab be an SBM, e.g. read in from read_SBM.read_SBM():

```
#compute edge scores in chunks of 1000 edges for each computation,
#multiprocessing with 4 processes
edgeprobs_mult = compute_single_edge_prob(sbm_metab,
        compute_all_spurious=True,
        compute_all_missing=True,
        disconn_nodes=[161],multiprocess_nodecount=4,
        maxedges_perjob=1000)
```

Note: Setting maxedges_perjob to 1000 has just been done here to show the output data structure in case of using chunking of edges. However, for this small example, this does not deliver an advantage in runtime - the overhead of starting a new process for each job will increase the runtime compared to launching only one process to compute all edge scores.

Example in which the scores are computed for a list of edges: Let sbm_metab be an SBM, e.g. read in from read_SBM.read_SBM(), and using two lists of missing edges for which to compute the edge probabilities, for example generated from the R files to determine missing edge lists:

```
miss_edge_list_conn = calcedge.read_in_edge_list_from_file("example_files/metab_
↪example_missing_edge_list_connected_nodes.txt")
miss_edge_list_disconn = calcedge.read_in_edge_list_from_file("example_files/
↪metab_example_missing_edge_list_disconnected_nodes.txt")
edgeprobs = calcedge.compute_single_edge_prob(sbm_metab,
        compute_all_spurious=False,
        compute_all_missing=False,
        disconn_nodes=[161],
        miss_edge_list=[miss_edge_list_conn,miss_edge_list_disconn],
        multiprocess_nodecount=4,maxedges_perjob=100)
```

calc_edge_prob.**determine_spur_edge_list**(*SBM_in*, *edge_covcrit*)

Function to determine the list of existing edges for which to compute edge confidence scores.

Parameters

- SBM_in: SBM of which to determine the edges
- **edge_covcrit: dict containing**
  - 'method': one of
    - 'all' (all spurious edges - default),
    - 'threshold' (up to a certain threshold of some other variable)
    - 'edge_count' (up to a certain number of edges)
  - if 'method'=='all' (this is the default), no further info required
  - if method=='threshold', we need two more fields

        * 'threshold_values' giving a vector of the values at which to set the threshold (in order of edges of SBM), and
        * 'cut_off' giving the actual cut-off (edges with threshold_values bigger than cut_off are used)
      - if method=='edge_count', we need two more fields
        * 'threshold_values' giving a vector of the values which to use to rank the edges, and
        * 'edge_count_value': an integer giving the number of edges to export (the edges with highest values are used)

    Returns

An edge list which can be used as input parameter 'spur_edge_list' to the function compute_single_edge_prob()

calc_edge_prob.**read_in_edge_list_from_file**(*filename*)

Function to read in a missing edge list as generated from the R script 'determine_missing_edge_lists.R' and bring it in Python usable format (zipping).

Parameters

- filename: the name of a file in which the node index pairs are assigned

Returns

An edge list in the format to use as input for the parameters 'spur_edge_list' or 'miss_edge_list' to the function compute_single_edge_prob()

Example

Read in an edge list from a tab-separated file:

```
miss_edge_list = read_in_edge_list_from_file('181203_test_edgelist_missconn_metab.
↪txt')
```

global_fit.**global_fit**(*ini_graph*, *random_ini_no*, *return_fit_no*, *model_opt*, *fit_method_opt*)

Perform a global fit of the initial graph on a certain SBM version. Wrapper for minimize_nested_blockmodel() and minimize_blockmodel().

Parameters

- ini_graph: initial graph to fit to SBM, Graph object, assumes that 'edge_weight' exists as edge property which can be fitted to
- random_ini_no: (optional) number of random initiations of the fit, default: 1
- return_fit_no: (optional) number of best fits to return, default: 1
- model_opt: (optional) Options for the SBM to use
  - 'dc_SBM' - Boolean, True [default] for degree corrected SBM
  - 'hierarchical_SBM' - Boolean, default False (planar, non-nested model), True for hierarchical (nested) SBM
  - 'overlap_SBM' - Boolean, default False, True if overlap should be computed
  - 'weighted_SBM' - Boolean, default False, True for weighted. NOTE: only the edge property 'edge_weight' is going to be incorporated as edge covariate
  - 'weight_prior': mandatory if 'weighted_SBM' == True, default: 'real-normal'; prior distribution assumed for the edge weight covariate, should be one of

        * 'real-exponential'
        * 'real-normal'
        * 'discrete-geometric'
        * 'discrete-binomial'
        * 'discrete-poisson'
- fit_method_opt: (optional) options for the global fit
  - 'method_string'
    * 'best_state' [default] - report the x best fits (lowest minimal description length, default graph-tool parameters), or
    * 'with_mcmc' - also perform an additional mcmc-equilibration for the best fits (not supported for nested overlapping SBM)
  - 'mcmc_fit_no' - integer giving the number of fits for which to equilibrate via MCMC sampling, default: return_fit_no
  - 'mcmc_niter' - number of iterations, default: 10
  - 'mcmc_nbreaks' - number of record-breaking events, default: 2
  - 'mcmc_wait' - number of cycles to wait without record-breaking event, default: 1000

Returns

- Three objects if fit_method_opt['method_string']=='best_state':
  - optimal minimal description lengths as list, number according to return_fit_no (not sorted!)
  - block state partition for these minimal description lengths, top_block_no (in the same order as the first return value)
  - all minimal description lengths in order of computation, number: random_ini_no
- Four objects if fit_method_opt['method_string']=='with_mcmc':
  - optimal minimal description lengths as list, number according to return_fit_no (not sorted!)
  - block state partition for these minimal description lengths, top_block_no (in the same order as the first return value)
  - all minimal description lengths (sorted) before applying MCMC sampling anew, number: random_ini_no
  - all minimal description lengths after the MCMC sampling, number: mcmc_fit_no (if this is not specified, return_fit_no)

Examples

Perform the model fit to a degree-corrected, hierarchical, non-overlapping, non-weighted SBM for 10 initial fits and return the partitions for the 2 best fits. Without additional MCMC fitting, we have 3 return values (best 2 minimal description lengths (not sorted!), partitions for the best 2 fits, minimal description lengths from all 5 initialized fits):

```
#import the initial graph
import read_graph as rg
graph_ini,ew = rg.read_graph('180905_metabbud_erneg_spear_scalefree_el.csv',total_
↪node_count = 162)
```

(continues on next page)

(continued from previous page)

```
#perform fit
import global_fit as gf
top_mindl0, opt_part0, all_mindl0 = gf.global_fit(graph_ini,random_ini_no = 5,
↪return_fit_no = 2,
        model_opt={'dc_SBM':True,'hierarchical_SBM':True, 'overlap_SBM':False,
↪'weighted_SBM':False},
        fit_method_opt={'method_string':'best_state'})
```

Second example: Model fit to non-degree-corrected, non-hierarchical, non-overlapping, weighted SBM (with real-normal prior) for 6 initial fits, performs additional MCMC sampling (with default settings) for the 4 best fits, and returns the partitions for the 2 best fits. With this additional MCMC sampling, we have 4 return values - the three from the first example plus the minimal description lengths from the 4 additional MCMC samplings:

```
top_mindl1, opt_part1, all_mindl1, mcmc_mindl1 = gf.global_fit(
        graph_ini,random_ini_no = 6,return_fit_no = 2,
        model_opt={'dc_SBM':False,'hierarchical_SBM':False, 'overlap_SBM':False,
↪'weighted_SBM':True,'weight_prior':'real-normal'},
        fit_method_opt={'method_string':'with_mcmc','mcmc_fit_no':4})
```

read_graph.**read_graph**(*edge_list_file_name*, *total_node_count=None*)

Establish a graph from an edge list including edge covariates.

The filename is used as input. In .csv files, a column 'edge_weight' is converted to float covariate. All edge covariates not named 'edge_weight' are assigned string datatypes. Nodes without edges, i.e. not ocurring in the edge list, are added (as last nodes) such that the node count of the graph coincides with total_node_count. If having .xml/.graphml/.gt as input, the file is just loaded as network and nodes without edges are added such that the number of nodes is correct. Note: We do not allow to have commas, spaces or '' within the edge property names. Also, no edge property should be named edge_index as this interferes with the corresponding graph-tool in-built edge property.

Parameters

- edge_list_file_name: a string giving the location of the file including its extension
- total_node_count: a positive integer giving the number of nodes the network should have. Omit or set to None if no nodes without edges should be added

Returns

- the network as Graph object
- a Boolean value indicating whether there is an edge_weight edge property

Example

Create a graph from the edge list provided in the example_files folder:

```
import read_graph as rg
graph_ini,ew = rg.read_graph('example_files/180905_metabbud_erneg_spear_scalefree_
↪el.csv',total_node_count = 162)
```

read_SBM.**read_SBM**(*edgelist_filename*, *total_node_count*, *file_name_block_list_start*, *file_name_block_list_end*, *no_hier_levels=1*, *is_hierarchical=True*, *is_degree_corrected=True*)

Create an SBM from a graph given by an edge list and the (hierarchical) partitions given in the files.

No additional graph characteristics are implemented, i.e. no edge or node property maps.

Parameters

- edgelist_filename: name of the edgelist of the underlying graph
- total_node_count: number of nodes which should be in the graph (disconnected nodes are added)
- file_name_block_list_start: beginning of filename before running index (assumed to start with 0)
- file_name_block_list_end: end of filename after running index
- no_hier_levels: number of hierarchy levels, defaults to 1 (planar SBM)
- is_hierarchical: whether the hierarchical or planar SBM is to be established
- is_degree_corrected: whether the SBM is degree-corrected (default) or not

Returns

A graph_tool::NestedBlockState or graph_tool::BlockState object with the graph from the edge list as underlying graph and the partition as given from the files.

Example

Using the example files to create an SBM:

```
import read_SBM as rsbm
sbm_metab = read_SBM("example_files/180905_metabbud_erneg_spear_scalefree_el.csv",
        162,
        "example_files/metabbud_erneg_sf_hSBM_bestfit_block_lev",
        ".txt",
        no_hier_levels = 3,
        is_hierarchical=True,
        is_degree_corrected=False)
```

read_SBM.**read_block_partition**(*file_name_block_list_start*, *file_name_block_list_end*, *no_hier_levels=1*)

Reads in block partition files for SBM construction. It relies on partition files in which the partition indices of each node will be given, e.g. as generated with write_fit_to_file.py. For the file naming, it is required that there is a running index (on the hierarchy level), starting from zero (lowest hierarchy level). The function output can be used to initiate an SBM with the according partition.

Parameters

- file_name_block_list_start: beginning of filename before running index
- file_name_block_list_end: end of filename after running index
- no_hier_levels: number of hierarchy levels, defaults to 1 (planar SBM)

Returns

list of np arrays which can used as partition input for SBM building

Example

Read in a hierarchical partition as given in the folder example_files:

```
import read_SBM as rsbm
block_prot = rsbm.read_block_partition("example_files/metabbud_erneg_sf_hSBM_
↪bestfit_block_lev",
        ".txt",no_hier_levels = 3)
```

write_fit_to_file.**write_fit_to_file**(*topmindl*, *topblockno*, *mindl*, *mcmc_mindl*, *filename*, *blockno_opt*)

Writes the data from global_fit into files.

Attention: Existing files of the same names are overwritten without warning. The files will end by

- '_top_mindl.txt' for the best fits
- '_mindl.txt' for all initial fits
- '_mcmc_mindl.txt' for the fits after additional MCMC sampling,
- '_top_block_no.txt' for the best partitions, number given in blockno_opt

The partition filename string will contain 'optk' for the kth best fit supplied to the function, it will contain 'runindk' with k indicating the position of this fit in the _top_mindl.txt file, it will contain 'levk' with k indicating the hierarchy level of the partition. For example, lev0 marks the partition of the nodes from the initial graph into blocks, lev1 marks the partition of the first-level blocks into blocks etc. The last level cumulates in the trivial partition of one top block.

Parameters

- topmindl: best minimal description lengths as returned as first variable from global_fit()
- topblockno: block partitions, in the same order as topmindl, as returned from global_fit()
- mindl: all ever computed minimal description lengths, returned as third variable from global_fit()
- mcmc_mindl: (optional) all minimal description lengths after additional MCMC sampling (4th output from global_fit())
- filename: how the written files should be named (except for their conserved ending), without extension
- blockno_opt:
  - 'export_blockno': integer [default: 5] - how many block states should be saved as files, all will be written if integer is larger than number of provided partitions
  - 'hierarchical_SBM': Boolean [default: False] - if non-nested SBM (False) or nested SBM (True) was used

Returns

No return value.

Examples

For a fit without MCMC sampling, hierarchical SBM:

```
import write_fit_to_file as wf
wf.write_fit_to_file(topmindl=top_mindl0,
        topblockno=opt_part,
        mindl=all_mindl,
        mcmc_mindl=None,
        filename="example_output_file0",
        blockno_opt={'export_blockno':2,'hierarchical_SBM':True})
```

For a fit with additional MCMC sampling, non-hierarchical SBM (default option):

```
wf.write_fit_to_file(top_mindl1,
opt_part1,
all_mindl1,
mcmc_mindl1,
```

(continues on next page)

(continued from previous page)

```
filename="example_output_file1",
blockno_opt={'export_blockno':2})
```

PYTHON MODULE INDEX

c

calc_edge_prob, 1

g

global_fit, 3

r

read_graph, 5
read_SBM, 5

w

write_fit_to_file, 6

INDEX

C

calc_edge_prob (*module*), 1
compute_single_edge_prob() (*in module calc_edge_prob*), 1

D

determine_spur_edge_list() (*in module calc_edge_prob*), 2

G

global_fit (*module*), 3
global_fit() (*in module global_fit*), 3

R

read_block_partition() (*in module read_SBM*), 6
read_graph (*module*), 5
read_graph() (*in module read_graph*), 5
read_in_edge_list_from_file() (*in module calc_edge_prob*), 3
read_SBM (*module*), 5
read_SBM() (*in module read_SBM*), 5

W

write_fit_to_file (*module*), 6
write_fit_to_file() (*in module write_fit_to_file*), 6