trisbm

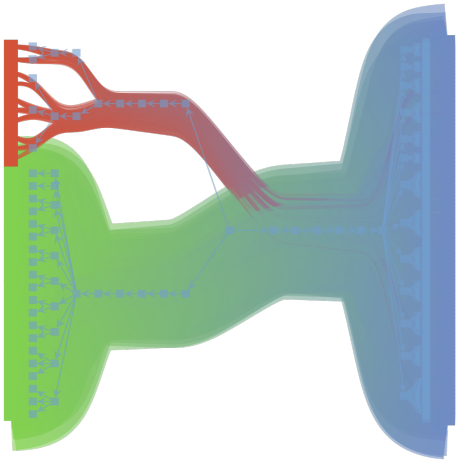
Release 0.0.1

Filippo Valle

Jul 05, 2022

CONTENTS:

This module inherits from sbmtm and extends network based topic models with **multiple layers** of information.



CHAPTER

ONE

TRISBM MODULE

install: *conda install nsbm -c conda-forge*

1.1 triSBM

triSBM

class trisbm.trisbm.**trisbm**

Class to run trisbm

_get_shape()

Returns
list of tuples (number of documents, number of words, (number of keywords,...))

clusters(*l=0*, *n=10*)

Get n 'most common' documents from each document cluster. most common refers to largest contribution in group membership vector. For the non-overlapping case, each document belongs to one and only one group with prob 1.

clusters_query(*doc_index*, *l=0*)

Get all documents in the same group as the query-document. Note: Works only for non-overlapping model. For overlapping case, we need something else.

draw(**args*, ***kwargs*) → None

Draw the network

Parameters

- ***args** – positional arguments to pass to self.state.draw
- ****kwargs** – keyword argument to pass to self.state.draw

dump_model(*filename='trisbm.pkl'*)

Dump model using pickle

To restore the model:

import cloudpickle as pickle file=open("trisbm.pkl" ,"rb") model = pickle.load(file)

file.close()

fit(*n_init=5*, *verbose=True*, *deg_corr=True*, *overlap=False*, *parallel=True*, *B_min=3*, *B_max=None*, **args*, ***kwargs*) → None

Fit using minimize_nested_blockmodel_dl

Parameters

- **n_init** – number of initialisation. The best will be kept
- **verbose** – Print output
- **deg_corr** – use deg corrected model
- **overlap** – use overlapping model
- **parallel** – perform parallel moves
- ***args** – positional arguments to pass to gt.minimize_nested_blockmodel_dl
- ****kwargs** – keywords arguments to pass to gt.minimize_nested_blockmodel_dl

fit_overlap(*n_init=1*, *hierarchical=True*, *B_min=20*, *B_max=160*, *parallel=True*, *verbose=True*)

Fit the sbm to the word-document network. - hierarchical, bool (default: True). Hierarchical SBM or Flat SBM. Flat SBM not implemented yet. - Bmin, int (default:20): pass an option to the graph-tool inference specifying the minimum number of blocks.

get_D()

return number of doc-nodes == number of documents

get_N()

return number of edges == tokens

get_V()

return number of word-nodes == types

get_groups(*l=0*)

return groups

Parameters
l – hierarchy level

get_mdl()

Get minimum description length

Proxy to self.state.entropy()

group_membership(*l=0*)

Return the group-membership vectors for

- document-nodes, p_td_d, array with shape Bd x D
- word-nodes, p_tw_w, array with shape Bw x V

It gives the probability of a nodes belonging to one of the groups.

group_to_group_mixture(*l=0*, *norm=True*)

load_graph(*filename='graph.xml.gz'*) → None

Load a presaved graph

Parameters
filename – graph to load

load_model(*filename='topsbm.pkl'*)

make_graph(*df: DataFrame*, *get_kind*) → None

Create a graph from a pandas DataFrame

Parameters

- **df** – DataFrame with words on index and texts on columns. Actually this is a BoW.
- **get_kind** – function that returns 1 or 2 given an element of df.index. [1 for words 2 for keywords]

make_graph_from_BoW_df(*df*, *counts=True*, *n_min=None*)

Load a graph from a Bag of Words DataFrame

:param : :type : type df: DataFrame should be a DataFrame with where df.index is a list of words and df.columns a list of documents :param optional arguments: :param - counts: :type - counts: save edge-multiplicity as counts (default: True) :param - n_min: :type - n_min: filter all word-nodes with less than n_min counts (default None) :param int: :type int: filter all word-nodes with less than n_min counts (default None)

make_graph_multiple_df(*df: DataFrame*, *df_keyword_list: list*) → None

Create a graph from two dataframes one with words, others with keywords or other layers of information

Parameters

- **df** – DataFrame with words on index and texts on columns
- **df_keyword_list** – list of DataFrames with keywords on index and texts on columns

metadata(*l=0*, *n=10*, *kind=2*)

get the n most common keywords for each keyword-group in level l.

Returns
tuples (keyword,P(kw|tk))

metadatumdist(*doc_index*, *l=0*, *kind=2*)

multiflip_mcmc_sweep(*n_steps=1000*, *beta=inf*, *niter=10*, *verbose=True*)

Fit the sbm to the word-document network. Use multtiplip_mcmc_sweep - n_steps, int (default:1): number of steps.

plot(*filename=None*, *nedges=1000*)

Plot the graph and group structure. optional: - filename, str; where to save the plot. if None, will not be saved - nedges, int; subsample to plot (faster, less memory)

plot_topic_dist(*l*)

print_summary(*tofile=True*)

Print hierarchy summary

print_topics(*l=0*, *format='csv'*, *path_save=''*)

Print topics, topic-distributions, and document clusters for a given level in the hierarchy.

Parameters

- **l** – level to store
- **format** – csv (default) or html
- **path_save** – path/to/store/file

save_data()

save_graph(*filename='graph.xml.gz'*) → None

Save the graph

Parameters

filename – name of the graph stored

search_consensus(*force_niter=100000*, *niter=100*)

topicdist(*doc_index*, *l=0*)

topics(*l=0*, *n=10*)

get the n most common words for each word-group in level l. return tuples (word,P(w|tw))

CHAPTER

TWO

LICENCE

This work is in part based on sbmtm and it is released under the terms of the GNU General Public License available along with this program or at <https://www.gnu.org/licenses/>

PYTHON MODULE INDEX

t

INDEX