trisbm

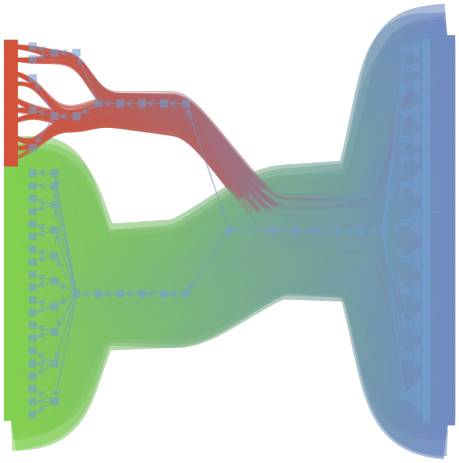
Release 0.5.1

Filippo Valle

Mar 31, 2023

CONTENTS:

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



Run multi-partite topic models.

CHAPTER

ONE

NSBM MODULE

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

1.1 Install

1.1.1 Install from conda-forge

To install nsbmm from Anaconda use the following command

conda install nsbm -c conda-forge

1.1.2 Build from source

Or you can build it from source

Dependencies:

- graph-tool > 2.40
- numpy
- matplotlib
- cloudpickle
- pandas

python3 -m pip install . -vv

1.2 nsbm

triSBM

You should have received a copy of the GNU General Public License along with this program. If not, see < http: // www.gnu.org/licenses/>.

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.

Parameters

- **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()

Returns
number of doc-nodes == number of documents

get_N()

Returns
number of edges == tokens

get_V()

Returns
number of word-nodes == types

get_groups(*l=0*)

Parameters
l – hierarchy level

Returns
groups

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

Parameters

- **df** – DataFrame should be a DataFrame with where df.index is a list of words and df.columns a list of documents
- **counts** – save edge-multiplicity as counts (default: True)
- **n_min** – 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

Parameters
n_steps – int (default:1): number of steps.

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

Plot the graph and group structure.

Parameters

- **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))

1.3 sbmtm

This module is cloned from https://github.com/martingerlach/hSBM_Topicmodel/commit/261d870cfc884c4f23ddaa213d07ccbddf348c78

class trisbm.trisbm.**sbmtm**

Class for topic-modeling with sbm's.

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.

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

fit(*overlap=False*, *hierarchical=True*, *B_min=2*, *B_max=None*, *n_init=1*, *parallel=False*, *verbose=False*)

Fit the sbm to the word-document network.

Parameters

- **overlap** – bool (default: False). Overlapping or Non-overlapping groups. Overlapping implemented in fit_overlap
- **hierarchical** – bool (default: True). Hierarchical SBM or Flat SBM. Flat SBM not implemented yet.
- **Bmin** – int (default:None): pass an option to the graph-tool inference specifying the minimum number of blocks.

- **n_init** – int (default:1): number of different initial conditions to run in order to avoid local minimum of MDL.
- **parallel** – passed to mcmc_sweep If parallel == False each vertex move attempt is made sequentially, where vertices are visited in random order. Otherwise the moves are attempted by sampling vertices randomly, so that the same vertex can be moved more than once, before other vertices had the chance to move.

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.

Parameters

- **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()

Returns
number of doc-nodes == number of documents

get_N()

Returns
number of edges == tokens

get_V()

Returns
number of word-nodes == types

get_groups(*l=0*)

extract statistics on group membership of nodes form the inferred state.

Parameters

- **B_d** – int, number of doc-groups
- **B_w** – int, number of word-groups
- **p_tw_w** – array B_w x V; word-group-membership: prob that word-node w belongs to word-group tw: P(tw | w)
- **p_td_d** – array B_d x D; doc-group membership: prob that doc-node d belongs to doc-group td: P(td | d)
- **p_w_tw** – array V x B_w; topic distribution: prob of word w given topic tw P(w | tw)
- **p_tw_d** – array B_w x d; doc-topic mixtures: prob of word-group tw in doc d P(tw | d)

Returns
dictionary

get_mdl()

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.gt.gz'*)

Load a word-document network generated by make_graph() and saved with save_graph().

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

make_graph(*list_texts*, *documents=None*, *counts=True*, *n_min=None*)

Load a corpus and generate the word-document network

optional arguments:

Parameters

- **documents** – list of str, titles of documents
- **counts** – save edge-multiplicity as counts (default: True)
- **n_min** – int filter all word-nodes with less than n_min counts (default None)

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

Load a graph from a Bag of Words DataFrame

Parameters

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

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

Fit the sbm to the word-document network. Use multtiplip_mcmc_sweep

Parameters
n_steps – int (default:1): number of steps.

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

Plot the graph and group structure.

Parameters

- **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
format – csv (default) or html

save_data()

save_graph(*filename='graph.gt.gz'*)

Save the word-document network generated by make_graph() as filename. Allows for loading the graph without calling make_graph().

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))

1.4 Tutorial

1.4.1 Create a fake dataset

```
from nsbm import nsbm
import pandas as pd
import numpy as np
df = pd.DataFrame(
    index=["w{}".format(w) for w in range(1000)],
    columns=["doc{}".format(d) for d in range(250)],
    data=np.random.randint(1, 100, 250000).reshape((1000, 250)))
df_key_list = []
# an additional feature
df_key_list.append(
    pd.DataFrame(
        index=["keyword{}".format(w) for w in range(100)],
        columns=["doc{}".format(d) for d in range(250)],
        data=np.random.randint(1, 10, (100, 250)))
)

# another additional feature
df_key_list.append(
    pd.DataFrame(
        index=["author{}".format(w) for w in range(10)],
        columns=["doc{}".format(d) for d in range(250)],
        data=np.random.randint(1, 5, (10, 250)))
)

# other features
df_key_list.append(
    pd.DataFrame(
        index=["feature{}".format(w) for w in range(25)],
        columns=["doc{}".format(d) for d in range(250)],
        data=np.random.randint(1, 5, (25, 250)))
)
```

- *df* is a Bag of Words (BoW) representation of the documents.
- *df_key_list* is a list of (BoW), all of them have to share the same columns (**documents**) in this case *keywords*, *authors* and *features* are the additional (more than words) information about the documents.

1.4.2 Create and fit a model

Create a model

```
model = nsbm()
model.make_graph_multiple_df(df, df_key_list)
```

Fit the model

```
model.fit(n_init=1, B_min=50, verbose=False)
```

Parameters:

- n_init the number of initializations: olny the one with the shortest DL will be kept
- B_min minimum number of blocks
- B_max maximum number of blocks
- parallel the model will be fitted with heavy parallelization
- verbose if True, print the progress

The fit is performed using `graph_tool.inference.minimize_nested_blockmodel_dl()`

Save the results

```
model.save_data()
```

1.4.3 Stochastic Block Models on graph_tool

For a complete tutorial on how to infer network structure using stochastic block models see graph_tool tutorial

1.5 License

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/>

Stochatic Block Model Topic Model (hSBM) was introduced by:

Gerlach, M et al. (**2018**). A network approach to topic models, *Science Advances*. https://doi.org/10.1126/sciadv.aaq1360

1.6 Cite this work

If you use, modify or extend this work please cite:

Valle, F. (**2021**). nsbm (Version 1.0.0), *Zenodo*. https://doi.org/10.5281/zenodo.5045445

```
@software{valle_nsbm_2021,
    author = {Valle, Filippo},
    doi = {10.5281/zenodo.5045445},
    month = {6},
    title = {{nsbm}},
    url = {https://github.com/fvalle1/nsbm},
    version = {1.0.0},
    year = {2021}
}
```

and the related works using transcriptomics data:

Valle, F. et al. (**2022**). Multiomics Topic Modeling for Breast Cancer Classification, *Cancers*. https://doi.org/10.3390/cancers14051150

Valle, F. et al. (**2020**). A Topic Modeling Analysis of TCGA Breast and Lung Cancer Transcriptomic Data, *Cancers*. https://doi.org/10.3390/cancers12123799

```
@article{valle_multiomics_2022,
    title = {Multiomics {Topic} {Modeling} for {Breast} {Cancer} {Classification}},
    volume = {14},
    issn = {2072-6694},
    url = {https://www.mdpi.com/2072-6694/14/5/1150},
    doi = {10.3390/cancers14051150},
    number = {5},
    journal = {Cancers},
    author = {Valle, Filippo and Osella, Matteo and Caselle, Michele},
    month = feb,
    year = {2022},
    pages = {1150},
}

@article{valle_topic_2020,
    title = {A {Topic} {Modeling} {Analysis} of {TCGA} {Breast} and {Lung} {Cancer}
↪{Transcriptomic} {Data}},
    volume = {12},
    url = {https://www.mdpi.com/2072-6694/12/12/3799},
    doi = {10.3390/cancers12123799},
    language = {en},
    number = {12},
    urldate = {2021-05-20},
    journal = {Cancers},
    author = {Valle, Filippo and Osella, Matteo and Caselle, Michele},
    month = dec,
    year = {2020},
    pages = {3799},
}
```

PYTHON MODULE INDEX

t

INDEX

T