# topsbm Documentation

***Release 0.1***

**topsbm developers**

**Feb 05, 2019**

# Contents

Martin Gerlach, Tiago P. Peixoto, and Eduardo G. Altmann, "A network approach to topic models," Science Advances (2018)

Software ported to Scikit-learn format by the Sydney Informatics Hub at the University of Sydney.

CHAPTER 1

# Installation

The latest release can be installed from PyPi using:

```
$ pip install topsbm
```

Install the development version from GitHub using:

```
$ pip install https://github.com/Sydney-Informatics-Hub/topsbm/archive/master.zip
```

or by cloning the source code:

```
$ git clone https://github.com/Sydney-Informatics-Hub/topsbm
$ cd topsbm
$ pip install .
```

## 1.1 Installing dependencies

topsbm requires graph-tool to already be installed, as it cannot be installed with *pip*.

A simple way to install graph-tool and its dependencies is to use conda:

```
$ conda install -c conda-forge -c flyem-forge scikit-learn graph-tool pygobject cairo␣
↪gtk3
```

or simply:

```
$ git clone https://github.com/Sydney-Informatics-Hub/topsbm
$ cd topsbm
$ conda env create
```

## 1.2 Check your installation

Check the installation has worked with:

```
$ python -m topsbm.check_install
```

or run the full test suite:

```
$ pip install pytest
$ pytest --pyargs topsbm
```

# CHAPTER 2

# API Documentation

**class** topsbm.**TopSBM**(*n_init=1*, *min_groups=None*, *max_groups=None*, *weighted_edges=True*, *random_state=None*)

A Scikit-learn compatible transformer for hSBM topic models

**Parameters**

**n_init** [int, default=1] Number of random initialisations to perform in order to avoid a local minimum of MDL. The minimum MDL solution is chosen.

**min_groups** [int, default=None] The minimum number of word and docuent groups to infer. This is also a lower bound on the number of topics.

**max_groups** [int, default=None] The maximum number of word and docuent groups to infer. This also an upper bound on the number of topics.

**weighted_edges** [bool, default=True] When True, edges are weighted instead of adding duplicate edges.

**random_state** [None, int or np.random.RandomState] Controls randomization. See Scikit-learn's glossary.

Note that if this is set, the global random state of libcore will be affected, and the global random state of numpy will be temporarily affected.

### References

Martin Gerlach, Tiago P. Peixoto, and Eduardo G. Altmann, "A network approach to topic models,". Science Advances (2018)

**Attributes**

**graph_** [graph_tool.Graph] Bipartite graph between samples (the first *n_samples_* vertices) and features (the remaining vertices)

**state_** Inference state from graphtool

**n_levels_** [int] The number of levels in the inferred hierarchy of groups.

**groups_** [dict] Results of group membership from inference. Key is an integer, indicating the level of grouping (starting from 0). Value is a dict of information about the grouping which contains:

**B_d** [int] number of doc-groups

**B_w** [int] number of word-groups

**p_tw_d** [array of shape (B_w, d)] doc-topic mixtures: prob of word-group tw in doc d P(tw | d)

**p_td_d** [array of shape (B_d, n_samples)] doc-group membership: prob that doc-node d belongs to doc-group td: P(td | d)

**p_tw_w** [array of shape (B_w, n_features)] word-group-membership: prob that word-node w belongs to word-group tw: P(tw | w)

**p_w_tw** [array of shape (n_features, B_w)] topic distribution: prob of word w given topic tw P(w | tw)

Here "d"/document refers to samples; "w"/word refers to features.

**mdl_** minimum description length of inferred state

**n_features_** [int]

**n_samples_** [int]

### Methods

| | |
|---|---|
| *fit*(X[, y]) | Fit the hSBM topic model |
| *fit_transform*(X[, y]) | Fit the hSBM topic model |
| get_params([deep]) | Get parameters for this estimator. |
| *plot_graph*([filename, n_edges]) | Plots arcs from documents to words coloured by inferred group |
| set_params(**params) | Set the parameters of this estimator. |

**fit** (*X*, *y=None*)
Fit the hSBM topic model

Constructs a graph representation of X and infers clustering.

**Parameters**

**X** [ndarray or sparse matrix of shape (n_samples, n_features)] Word frequencies for each document, represented as non-negative integers.

**y** [ignored]

**Returns**

**self**

**fit_transform** (*X*, *y=None*)
Fit the hSBM topic model

Constructs a graph representation of X, infers clustering, and reports the cluster probability for each sample in X.

**Parameters**

**X** [ndarray or sparse matrix of shape (n_samples, n_features)] Word frequencies for each document, represented as non-negative integers.

**y** [ignored]

**Returns**

**Xt** [ndarray of shape (n_samples, n_components)] The cluster probability for each sample in X

**plot_graph** (*filename=None*, *n_edges=1000*)
Plots arcs from documents to words coloured by inferred group

**Parameters**

**filename** [str, optional] Path to write to (e.g. 'something.png'). Otherwise returns a displayable object.

**n_edges** [int] Size of subsample to plot (reducing memory requirements)

# CHAPTER 3

# Example: Introduction to topsbm

Topic modelling with hierarchical stochastic block models

[1]:
```
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd
from topsbm import TopSBM
```

## 3.1 Setup: Load a corpus

1. We have a list of documents, each document contains a list of words.
2. We have a list of document titles (optional)

The example corpus consists of 63 articles from Wikipedia taken from 3 different categories (Experimental Physics, Chemical Physics, and Computational Biology).

We use scikit-learn's CountVectorizer to turn this text into a feature matrix.

[2]:
```
# Load texts and vectorize
with open('corpus.txt', 'r') as f:
    docs = f.readlines()

vec = CountVectorizer(token_pattern=r'\S+')
X = vec.fit_transform(docs)

# X is now a sparse matrix of (docs, words)

# titles corresponding to docs
with open('titles.txt', 'r') as f:
    x = f.readlines()
titles = [h.split()[0] for h in x]
```

[3]:
```
# view the data for document 0
print(titles[0])
print(docs[0][:100])
```

```
Nuclear_Overhauser_effect
 the nuclear overhauser effect noe is the transfer of nuclear spin polarization from␣
↪one nuclear spi
```

## 3.2 Fit the model

Calling `TopSBM.fit_transform` will: * construct the bipartite graph between documents and words (samples and features) * perform Hierarchical Stochastic Block Model inference over the graph * return an embedding of the samples in the block level with finest granularity

[18]:
```
model = TopSBM(random_state=9)
Xt = model.fit_transform(X)
```

## 3.3 Plotting the graph and block structure

The following plot shows the (hierarchical) community structure in the word-document network as inferred by the stochastic block model:

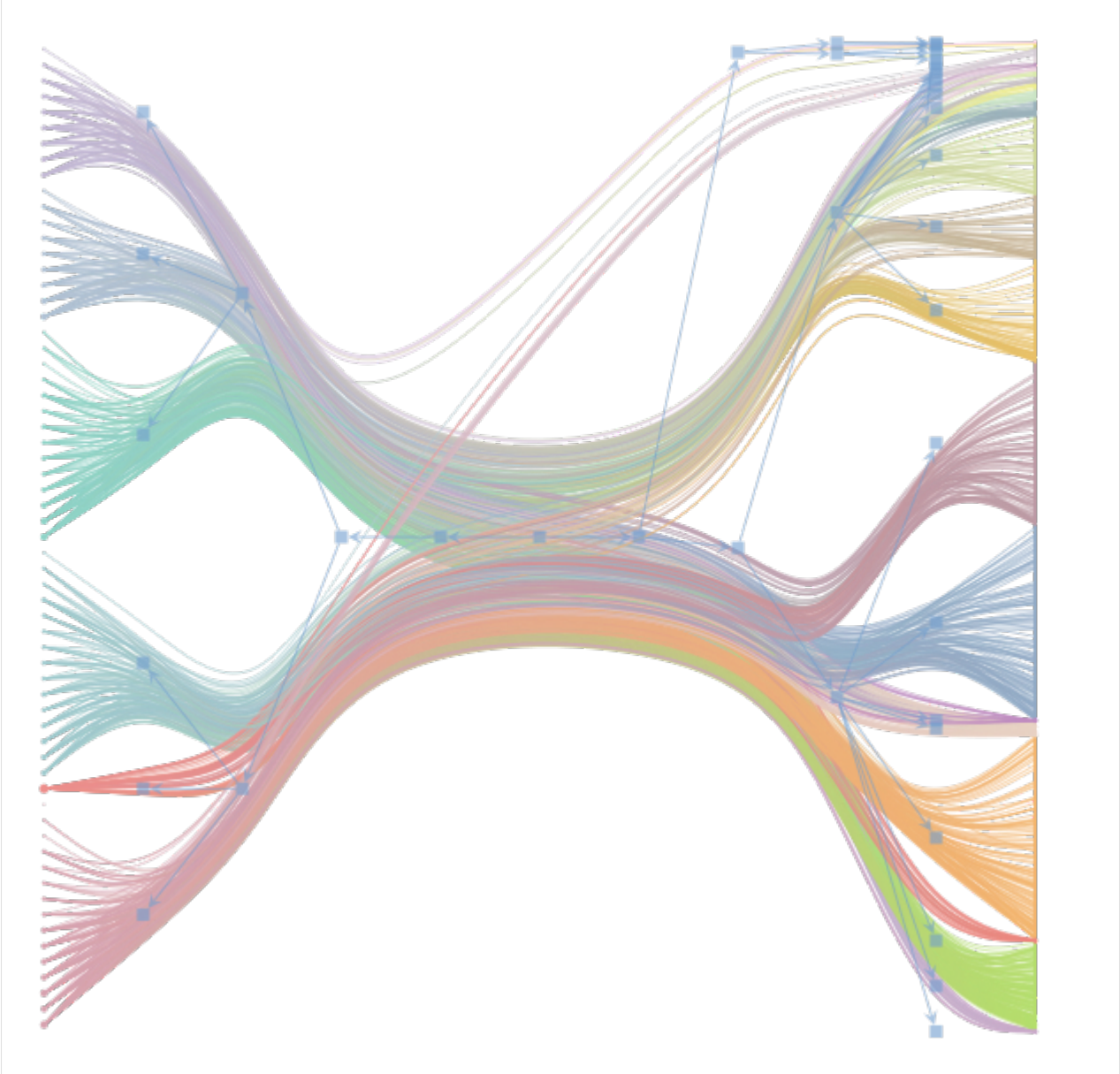
- document-nodes are on the left
- word-nodes are on the right
- different colors correspond to the different groups

The result is a grouping of nodes into groups on multiple levels in the hierarchy:

- on the uppermost level, each node belongs to the same group (square in the middle)
- on the next-lower level, we split the network into two groups: the word-nodes and the document-nodes (blue sqaures to the left and right, respectively). This is a trivial structure due to the bipartite character of the network.
- only next lower levels constitute a non-trivial structure: We now further divide nodes into smaller groups (document-nodes into document-groups on the left and word-nodes into word-groups on the right)

In the code, the lowest level is known as level 0, with coarser levels 1, 2, ...

[19]:
```
model.plot_graph(n_edges=1000)
```



## 3.4 Topics

For each word-group on a given level in the hierarchy, we retrieve the $n$ most common words in each group – these are the topics!

[20]:
```
topics = pd.DataFrame(model.groups_[1]['p_w_tw'],
                      index=vec.get_feature_names())
```

[21]:
```
for topic in topics.columns:
    print(topics[topic].nlargest(10))
    print()
```

```
the     0.006768
of      0.006661
a       0.006554
in      0.006446
is      0.006446
to      0.006339
and     0.006124
for     0.005372
an      0.005264
as      0.005264
Name: 0, dtype: float64

when        0.008921
where       0.008058
first       0.006619
given       0.006331
if          0.006331
field       0.006043
applied     0.005755
because     0.005755
e           0.005468
energy      0.005468
Name: 1, dtype: float64

computational    0.060606
development      0.055556
proteins         0.045455
open             0.040404
protein          0.040404
software         0.040404
community        0.035354
researchers      0.035354
core             0.025253
identify         0.025253
Name: 2, dtype: float64

point       0.217391
formula     0.188406
must        0.144928
wave        0.115942
spectrum    0.086957
air         0.072464
plane       0.057971
flow        0.043478
q           0.043478
mode        0.028986
Name: 3, dtype: float64
```

## 3.5 Topic-distribution in each document

Which level-1 topics contribute to each document?

[22]:
```
pd.DataFrame(model.groups_[1]['p_tw_d'],
             columns=titles)
```

[22]:
```
   Nuclear_Overhauser_effect  Quantum_solvent  Rovibrational_coupling  \
0                   0.608392            0.856                0.523529
1                   0.391608            0.144                0.458824
2                   0.000000            0.000                0.000000
3                   0.000000            0.000                0.017647

   Effective_field_theory  Chemical_physics  Rotational_transition  \
0                0.804651              0.82               0.648649
1                0.190698              0.16               0.337838
2                0.000000              0.00               0.000000
3                0.004651              0.02               0.013514

   Dynamic_nuclear_polarisation  Knight_shift  Polarizability  \
0                      0.584192      0.582418        0.493274
1                      0.412371      0.406593        0.500000
2                      0.000000      0.010989        0.002242
3                      0.003436      0.000000        0.004484

   Anisotropic_liquid               ...                \
0            0.645161               ...
1            0.354839               ...
2            0.000000               ...
3            0.000000               ...

   Louis_and_Beatrice_Laufer_Center_for_Physical_and_Quantitative_Biology  \
0                                           0.907692
1                                           0.092308
2                                           0.000000
3                                           0.000000

   Law_of_Maximum  Enzyme_Function_Initiative  SnoRNA_prediction_software  \
0        0.851351                    0.857143                    0.857143
1        0.121622                    0.095238                    0.142857
2        0.027027                    0.044218                    0.000000
3        0.000000                    0.003401                    0.000000

   Sepp_Hochreiter  Aureus_Sciences  \
0         0.846690         0.822222
1         0.139373         0.133333
2         0.013937         0.044444
3         0.000000         0.000000

   IEEE/ACM_Transactions_on_Computational_Biology_and_Bioinformatics  \
0                                            0.84375
1                                            0.09375
2                                            0.06250
3                                            0.00000

   Knotted_protein    BioUML  De_novo_transcriptome_assembly
0         0.773585  0.870647                        0.868932
1         0.169811  0.084577                        0.092233
2         0.047170  0.044776                        0.038835
3         0.009434  0.000000                        0.000000

[4 rows x 63 columns]
```

## 3.6 Extra: Clustering of documents - for free.

The stochastic block models clusters the documents into groups. We do not need to run an additional clustering to obtain this grouping.

For a query article, we can return all articles from the same group

[23]:
```
cluster_labels = pd.DataFrame(model.groups_[1]['p_td_d'],
                              columns=titles).idxmax(axis=0)
cluster_idx = cluster_labels['Rovibrational_coupling']
cluster_labels[cluster_labels == cluster_idx]
```

[23]:
```
Nuclear_Overhauser_effect                          0
Rovibrational_coupling                             0
Rotational_transition                              0
Dynamic_nuclear_polarisation                       0
Knight_shift                                       0
Polarizability                                     0
Anisotropic_liquid                                 0
Rotating_wave_approximation                        0
Molecular_vibration                                0
Fuel_mass_fraction                                 0
Electrostatic_deflection_(structural_element)      0
Magic_angle_(EELS)                                 0
Reactive_empirical_bond_order                      0
Photofragment-ion_imaging                          0
Molecular_beam                                     0
McConnell_equation                                 0
Ziff-Gulari-Barshad_model                          0
Empirical_formula                                  0
Newton's_laws_of_motion                            0
Ripple_tank                                        0
Particle-induced_X-ray_emission                    0
Elevator_paradox_(physics)                         0
Wave_tank                                          0
X-ray_crystal_truncation_rod                       0
Faraday_cup_electrometer                           0
Line_source                                        0
X-ray_standing_waves                               0
Point_source                                       0
Fragment_separator                                 0
Dynamic_mode_decomposition                         0
Euler's_laws_of_motion                             0
Quantum_oscillations_(experimental_technique)      0
dtype: int64
```

- genindex

# Python Module Index

## t

# Index