signet Documentation

Release 0.0.1

Peter Davies and Aldo Glielmo

Feb 19, 2021

Table of Contents

SigNet is a Python package for clustering of Signed Networks.

The code can be found on GitHub at https://github.com/alan-turing-institute/SigNet .

CHAPTER 1

Installation

1.1 Installing the requirements

This package is based on numpy, scipy, networkx, sklearn and cvxpy. These can be easily installed using anaconda or pip. Alternatively, they will be automatically installed with the package.

1.2 Installing the package

Install the latest version from the Github repository via

```
pip install git+https://github.com/alan-turing-institute/SigNet.git
```

CHAPTER 2

Implemented Algorithms

The algorithms currently implemented in the package can be *clustered* in three broad groups

2.1 Spectral clustering

These algorithms involve finding the top (lowest or highest) eigenvectors of a specific matrix. Depending on the matrix used, one can distinguish several algorithms. Some well known matrices which can be used for signed networks are:

- Adjacency
- Signed Laplacian matrix

2.2 Semidefinite clustering

These algorithms involve the solution of a semidefinite programming optimisation problem.

2.3 Generalised eigenproblem clustering

These algorithms involve the finding the top (lowest or highest) eigenvectors of a pair of matrices.

CHAPTER 3

Typical usage of the package

A typical usage of SigNet involves the initialisation of the Cluster class with a given pair of adjacency matrices and a subsequent application of a specific method.

```
from signet.cluster import Cluster
from signet.block_models import SSMB
from sklearn.metrics import adjusted_rand_score


# simple test on the signed stochastic block model

n = 50000  # number of nodes
k = 2      # number of clusters
eta = 0.1  # sign flipping probability
p = 0.0002 # edge probability

(Ap, An), true_assignment = SSBM(n = n, k = k, pin = p, etain = eta) # construct a␣
↪graph

c = Cluster((Ap, An))

predictions = c.spectral_cluster_laplacian(k = k, normalisation='sym') # cluster with␣
↪the signed laplacian
score = adjusted_rand_score(predictions, true_assignment)

print(score)
```

CHAPTER 4

Modules (API reference)

The package contains three modules: cluster, block_models and utils.

4.1 The cluster module

Inside the cluster module one can find the Cluster class, which is the main class of the package. It is inisialised with a pair of adjacency matrices (one for the positive and one for the negative graph) and it contain all the implemented algorithms as class methods.

class `cluster.`**`Cluster`**(*data*)

Class containing all clustering algorithms for signed networks.

This should be initialised with a tuple of two csc matrices, representing positive and negative adjacency matrix respectively (A^+ and A^-). It contains clustering algorithms as methods and graph specifications as attributes.

Parameters **`data`** (*`tuple`*) – Tuple containing positive and negative adjacency matrix (A^+, A^-).

`p`

positive adjacency matrix.

Type csc matrix

`n`

negative adjacency matrix.

Type csc matrix

`A`

total adjacency matrix.

Type csc matrix

`D_p`

diagonal degree matrix of positive adjacency.

Type csc matrix

`D_n`
diagonal degree matrix of negative adjacency.

Type csc matrix

`Dbar`
diagonal signed degree matrix.

Type csc matrix

`normA`
symmetrically normalised adjacency matrix.

Type csc matrix

`size`
number of nodes in network

Type int

`SDP_cluster`(*k*, *solver='BM_proj_grad'*, *normalisation='sym_sep'*)
Clustering based on a SDP relaxation of the clustering problem.

A low dimensional embedding is obtained via the lowest eigenvectors of positive-semidefinite matrix Z which maximises its Frobenious product with the adjacency matrix and k-means is performed in this space.

Parameters

- **`k`** (*`int, or list of int`*) – The number of clusters to identify. If a list is given, the output is a corresponding list.
- **`solver`** (*`str`*) – Type of solver for the SDP formulation. 'interior_point_method' - Interior point method. 'BM_proj_grad' - Burer Monteiro method using projected gradient updates. 'BM_aug_lag' - Burer Monteiro method using augmented Lagrangian updates.

Returns Label assignments.

Return type array of int, or list of array of int

`SPONGE`(*k=4*, *tau_p=1*, *tau_n=1*, *eigens=None*, *mi=None*)
Clusters the graph using the Signed Positive Over Negative Generalised Eigenproblem (SPONGE) clustering.

The algorithm tries to minimises the following ratio (Lbar^+ + tau_n D^-)/(Lbar^- + tau_p D^+). The parameters tau_p and tau_n can be typically set to one.

Parameters

- **`k`** (*`int, or list of int`*) – The number of clusters to identify. If a list is given, the output is a corresponding list.
- **`tau_n`** (*`float`*) – regularisation of the numerator
- **`tau_p`** (*`float`*) – regularisation of the denominator

Returns Output assignment to clusters.

Return type array of int, or list of array of int

Other Parameters

- **eigens** (*int*) – The number of eigenvectors to take. Defaults to k.
- **mi** (*int*) – The maximum number of iterations for which to run eigenvlue solvers. Defaults to number of nodes.
- **nudge** (*int*) – Amount added to diagonal to bound eigenvalues away from 0.

SPONGE_sym(*k=4*, *tau_p=1*, *tau_n=1*, *eigens=None*, *mi=None*)

Clusters the graph using the symmetric normalised version of the SPONGE clustering algorithm.

The algorithm tries to minimises the following ratio (Lbar_sym^+ + tau_n Id)/(Lbar_sym^- + tau_p Id). The parameters tau_p and tau_n can be typically set to one.

Parameters

- **k** (*int, or list of int*) – The number of clusters to identify. If a list is given, the output is a corresponding list.
- **tau_n** (*float*) – regularisation of the numerator
- **tau_p** (*float*) – regularisation of the denominator

Returns Output assignment to clusters.

Return type array of int, or list of array of int

Other Parameters

- **eigens** (*int*) – The number of eigenvectors to take. Defaults to k.
- **mi** (*int*) – The maximum number of iterations for which to run eigenvlue solvers. Defaults to number of nodes.
- **nudge** (*int*) – Amount added to diagonal to bound eigenvalues away from 0.

find_eigenvalues(*k=100*, *matrix='laplacian'*)

Find top or bottom k eigenvalues of adjacency or laplacian matrix.

The list of the top (bottom) k eigenvalues of the adjacency (laplacian) matrix is returned. This can be useful in identifying the number of clusters.

Note: The Laplacian matrix used is the signed symmetric Laplacian.

Parameters

- **k** (*int*) – Number of eigenvalues to return
- **matrix** (*str*) – Type of matrix to diagonalise (either 'adjacency' or 'laplacian')

Returns

An array of the first k eigenvalues, ordered in ascending or descending order (depending on the matrix type)

Return type array of float

geproblem_adjacency(*k=4*, *normalisation='multiplicative'*, *eigens=None*, *mi=None*, *nudge=0.5*)

Clusters the graph by solving a adjacency-matrix-based generalised eigenvalue problem.

Parameters

- **k** (*int, or list of int*) – The number of clusters to identify. If a list is given, the output is a corresponding list.
- **normalisation** (*string*) – How to normalise for cluster size: 'none' - do not normalise. 'additive' - add degree matrices appropriately. 'multiplicative' - multiply by degree matrices appropriately.

Returns Output assignment to clusters.

Return type array of int, or list of array of int

Other Parameters

- **eigens** (*int*) – The number of eigenvectors to take. Defaults to k.
- **mi** (*int*) – The maximum number of iterations for which to run eigenvlue solvers. Defaults to number of nodes.
- **nudge** (*int*) – Amount added to diagonal to bound eigenvalues away from 0.

`geproblem_laplacian` (*k=4*, *normalisation='multiplicative'*, *eigens=None*, *mi=None*, *tau=1.0*)
Clusters the graph by solving a Laplacian-based generalised eigenvalue problem.

Parameters

- **`k`** (*`int, or list of int`*) – The number of clusters to identify. If a list is given, the output is a corresponding list.
- **`normalisation`** (*`string`*) – How to normalise for cluster size: 'none' - do not normalise. 'additive' - add degree matrices appropriately. 'multiplicative' - multiply by degree matrices appropriately.

Returns Output assignment to clusters.

Return type array of int, or list of array of int

Other Parameters

- **eigens** (*int*) – The number of eigenvectors to take. Defaults to k.
- **mi** (*int*) – The maximum number of iterations for which to run eigenvlue solvers. Defaults to number of nodes.
- **nudge** (*int*) – Amount added to diagonal to bound eigenvalues away from 0.

`spectral_cluster_adjacency` (*k=2*, *normalisation='sym_sep'*, *eigens=None*, *mi=None*)
Clusters the graph using eigenvectors of the adjacency matrix.

Parameters

- **`k`** (*`int, or list of int`*) – The number of clusters to identify. If a list is given, the output is a corresponding list.
- **`normalisation`** (*`string`*) – How to normalise for cluster size: 'none' - do not normalise. 'sym' - symmetric normalisation. 'rw' - random walk normalisation. 'sym_sep' - separate symmetric normalisation of positive and negative parts. 'rw_sep' - separate random walk normalisation of positive and negative parts.

Returns Output assignment to clusters.

Return type array of int, or list of array of int

Other Parameters

- **eigens** (*int*) – The number of eigenvectors to take. Defaults to k.
- **mi** (*int*) – The maximum number of iterations for which to run eigenvlue solvers. Defaults to number of nodes.

`spectral_cluster_adjacency_reg` (*k=2*, *normalisation='sym_sep'*, *tau_p=None*, *tau_n=None*, *eigens=None*, *mi=None*)
Clusters the graph using eigenvectors of the regularised adjacency matrix.

Parameters

- **`k`** (*`int`*) – The number of clusters to identify.

- **normalisation** (*string*) – How to normalise for cluster size: 'none' - do not normalise. 'sym' - symmetric normalisation. 'rw' - random walk normalisation. 'sym_sep' - separate symmetric normalisation of positive and negative parts. 'rw_sep' - separate random walk normalisation of positive and negative parts.
- **tau_p** (*int*) – Regularisation coefficient for positive adjacency matrix.
- **tau_n** (*int*) – Regularisation coefficient for negative adjacency matrix.

Returns Output assignment to clusters.

Return type array of int

Other Parameters

- **eigens** (*int*) – The number of eigenvectors to take. Defaults to k.
- **mi** (*int*) – The maximum number of iterations for which to run eigenvlue solvers. Defaults to number of nodes.

spectral_cluster_bethe_hessian (*k*, *mi=1000*, *r=None*, *justpos=True*)

Clustering based on signed Bethe Hessian.

A low dimensional embedding is obtained via the lowest eigenvectors of the signed Bethe Hessian matrix Hbar and k-means is performed in this space.

Parameters

- **k** (*int, or list of int*) – The number of clusters to identify. If a list is given, the output is a corresponding list.
- **mi** (*int*) – Maximum number of iterations of the eigensolver.
- **type** (*str*) – Types of normalisation of the Laplacian matrix. 'unnormalised', 'symmetric', 'random_walk'.

Returns Label assignments. int: Suggested number of clusters for network.

Return type array of int, or list of array of int

spectral_cluster_bnc (*k=2*, *normalisation='sym'*, *eigens=None*, *mi=None*)

Clusters the graph by using the Balance Normalised Cut or Balance Ratio Cut objective matrix.

Parameters

- **k** (*int, or list of int*) – The number of clusters to identify. If a list is given, the output is a corresponding list.
- **normalisation** (*string*) – How to normalise for cluster size: 'none' - do not normalise. 'sym' - symmetric normalisation. 'rw' - random walk normalisation.

Returns Output assignment to clusters.

Return type array of int, or list of array of int

Other Parameters

- **eigens** (*int*) – The number of eigenvectors to take. Defaults to k.
- **mi** (*int*) – The maximum number of iterations for which to run eigenvlue solvers. Defaults to number of nodes.

spectral_cluster_laplacian (*k=2*, *normalisation='sym_sep'*, *eigens=None*, *mi=None*)

Clusters the graph using the eigenvectors of the graph signed Laplacian.

Parameters

- **k** (*int, or list of int*) – The number of clusters to identify. If a list is given, the output is a corresponding list.
- **normalisation** (*string*) – How to normalise for cluster size: 'none' - do not normalise. 'sym' - symmetric normalisation. 'rw' - random walk normalisation. 'sym_sep' - separate symmetric normalisation of positive and negative parts. 'rw_sep' - separate random walk normalisation of positive and negative parts.

Returns Output assignment to clusters.

Return type array of int, or list of array of int

Other Parameters

- **eigens** (*int*) – The number of eigenvectors to take. Defaults to k.
- **mi** (*int*) – The maximum number of iterations for which to run eigenvlue solvers. Defaults to number of nodes.

waggle (*k*, *labs*, *matrix=None*, *rounds=50*, *mini=False*)

Postprocessing based on iteratively merging and cutting clusters of the provided solution.

Pairs of clusters are merged randomly. Merged clusters are then partitioned in two by spectral clustering on input matrix.

Parameters

- **k** (*int*) – The number of clusters to identify.
- **labs** (*array of int*) – Initial assignment to clusters.
- **matrix** (*csc matrix*) – Matrix to use for partitioning. Defaults to un-normalised adjacency matrix.

Returns Output assignment to clusters.

Return type array of int

Other Parameters

- **rounds** (*int*) – Number of iterations to perform.
- **mini** (*boolean*) – Whether to minimise (rather than maximise) the input matrix objective when partitioning.

4.2 The block_models module

This module contains a series of function that can generate random graphs with a signed community structure.

block_models.**SBAM** (*n*, *k*, *p*, *eta*)

A signed Barabási–Albert model graph generator.

Parameters

- **n** – (int) Number of nodes.
- **k** – (int) Number of communities.
- **p** – (float) Sparsity value.
- **eta** – (float) Noise value.

Returns (a,b),c where a is a sparse n by n matrix of positive edges, b is a sparse n by n matrix of negative edges c is an array of cluster membership.

`block_models.`**`SRBM`**(*n*, *k*, *p*, *eta*)
A signed regular graph model generator.

Parameters

- **`n`** – (int) Number of nodes.
- **`k`** – (int) Number of communities.
- **`p`** – (float) Sparsity value.
- **`eta`** – (float) Noise value.

Returns (a,b),c where a is a sparse n by n matrix of positive edges, b is a sparse n by n matrix of negative edges c is an array of cluster membership.

`block_models.`**`SSBM`**(*n*, *k*, *pin*, *etain*, *pout=None*, *etaout=None*, *values='ones'*, *sizes='uniform'*)
A signed stochastic block model graph generator.

Parameters

- **`n`** – (int) Number of nodes.
- **`k`** – (int) Number of communities.
- **`pin`** – (float) Sparsity value within communities.
- **`etain`** – (float) Noise value within communities.
- **`pout`** – (float) Sparsity value between communities.
- **`etaout`** – (float) Noise value between communities.
- **`values`** – (string) Edge weight distribution (within community and without sign flip; otherwise weight is negated): 'ones': Weights are 1. 'gaussian': Weights are Gaussian, with variance 1 and expectation of 1.# 'exp': Weights are exponentially distributed, with parameter 1. 'uniform: Weights are uniformly distributed between 0 and 1.
- **`sizes`** – (string) How to generate community sizes: 'uniform': All communities are the same size (up to rounding). 'random': Nodes are assigned to communities at random. 'uneven': Communities are given affinities uniformly at random, and nodes are randomly assigned to communities weighted by their affinity.

Returns (a,b),c where a is a sparse n by n matrix of positive edges, b is a sparse n by n matrix of negative edges c is an array of cluster membership.

4.3 The utils module

The utils module mainly contains functions that are used elsewhere in the package. It also contains the function *objscore* calculating the value of an arbitrary objective function on a given graph partition.

`utils.`**`cut`**(*elemlist*, *matrix*, *numbers*, *dc*, *mini*)
Cuts clusters by separately normalised PCA.

Parameters

- **`elemlist`** (*`list of lists of int`*) – Specifies the members of each cluster in the current clustering
- **`matrix`** (*`csc matrix`*) – Matrix objective with which to cut.
- **`numbers`** (*`list of int`*) – Marks previous clustering to use as starting vector.
- **`dc`** (*`boolean`*) – Whether to skip cutting last cluster

- **mini** (*boolean*) – Whether to minimise (instead of maximise) matrix objective.

Returns new cluster constituents

Return type list of lists of int

`utils.`**`invdiag`**(*M*)

Inverts a positive diagonal matrix.

Parameters **M** (*csc matrix*) – matrix to invert

Returns scipy sparse matrix of inverted diagonal

`utils.`**`merge`**(*elemlist*)

Merges pairs of clusters randomly.

Parameters **elemlist** (*list of lists of int*) – Specifies the members of each cluster in the current clustering

Returns New cluster constituents boolean: Whether last cluster was unable to merge list of int: List of markers for current clustering, to use as starting vectors.

Return type list of lists of int

`utils.`**`objscore`**(*labels*, *k*, *mat1*, *mat2=None*)

Scores a clustering using the objective matrix given

Parameters

- **labels** (*list of int*) – Clustering assignment.
- **k** (*int*) – Number of clusters.
- **mat1** (*csc matrix*) – Numerator matrix of objective score.
- **mat2** (*csc matrix*) – Denominator matrix of objective score. Default is no denominator.

Returns Score.

Return type float

`utils.`**`sqrtinvdiag`**(*M*)

Inverts and square-roots a positive diagonal matrix.

Parameters **M** (*csc matrix*) – matrix to invert

Returns scipy sparse matrix of inverted square-root of diagonal

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index