absorbing-centrality

Release

September 04, 2015

Contents

Contents:

CHAPTER 1

Overview

1.1 Absorbing Random-Walk Centrality

- Free software: ISC license

1.1.1 Installation

```
pip install absorbing_centrality
```

1.1.2 Documentation

https://absorbing-centrality.readthedocs.org/

1.1.3 Development

To run the all tests run:

```
tox
```

CHAPTER 2

Installation

At the command line:

```
pip install absorbing_centrality
```

CHAPTER 3

Usage

To use Absorbing Random-Walk Centrality in a project:

```
import absorbing_centrality
```

CHAPTER 4

Reference

4.1 Computing the centrality of a set

absorbing_centrality(G, team[, query, P, ...])	Compute the absorbing centrality of a team.
absorbing_centrality_inversion(G, team[, ...])	Compute the absorbing centrality of a team using a fast inversion with S

4.1.1 absorbing_centrality.absorbing_centrality

absorbing_centrality(*G*, *team*, *query=None*, *P=None*, *epsilon=1e-05*, *max_iterations=None*, *with_restarts=False*, *alpha=0.85*)

Compute the absorbing centrality of a team. The algorithm works by iteratively computing the powers of the non-absorbing submatrix of the transition matrix P.

Parameters

- **G** (*NetworkX graph*) – The graph on which to compute the centrality.
- **team** (*list*) – The team of nodes, whose centrality to compute.
- **query** (*list, optional*) – The set of query nodes to use for the random walks. If None (default) or empty, the query set is equal to the set of all nodes in the graph.
- **P** (*matrix, optional*) – The precomputed transition matrix of the graph (default is None).
- **epsilon** (*float, optional*) – The iterative algorithm stops when the error between the centrality computed by two successive iterations falls below epsilon (default is 1e-5).
- **max_iterations** (*int, optional*) – The upper limit to the number of iteratios of the algorithm (default is None).
- **with_restarts** (*bool, optional*) – If True, restarts the random surfer to the the query set (default is False).
- **alpha** (*float, optional*) – The probability of the random surfer to continue (default is 0.85).

Returns **score** – The absorbing centrality score.

Return type float

Note: Both *team* and *query* should use the original node names.

4.1.2 absorbing_centrality.absorbing_centrality_inversion

absorbing_centrality_inversion(*G*, *team*, *query=None*, *with_restarts=False*, *alpha=0.85*)
Compute the absorbing centrality of a team using a fast inversion with SuperLU solver.

Parameters

- **G** (*NetworkX graph*) – The graph on which to compute the centrality.
- **team** (*list*) – The team of nodes, whose centrality to compute.
- **query** (*list, optional*) – The set of query nodes to use for the random walks. If None (default) or empty, the query set is equal to the set of all nodes in the graph.
- **with_restarts** (*bool, optional*) – If True, restarts the random surfer to the the query set (default is False).
- **alpha** (*float, optional*) – The probability of the random surfer to continue (default is 0.85).

Returns **score** – The absorbing centrality score.

Return type float

Note: Both *team* and *query* should use the original node names.

4.2 Team-selection algorithms

greedy_team(G, k[, query, candidates, ...])	Selects a team of nodes according to the greedy algorithm.

4.2.1 absorbing_centrality.greedy_team

greedy_team(*G*, *k*, *query=None*, *candidates=None*, *fast_select=False*, *return_times=False*, *with_restarts=False*, *alpha=0.85*)
Selects a team of nodes according to the greedy algorithm.

Parameters

- **G** (*Networkx graph*) – The graph from which the team will be selected.
- **k** (*int*) – The size of the team.
- **query** (*list, optional*) – If provided, the distance is measured with respect to the nodes in *query*.
- **candidates** (*list, optional*) – If provided, the team is picked only among the nodes in *candidates*.
- **fast_select** (*bool, optional*) – If True, the greedy algorithm will only consider candidates in a smart way, by examining their gain at each round (default is False).
- **with_restarts** (*bool, optional*) – If True, the greedy algorithm is based on the transition matrix w/ restarts to the supernode (default is False).
- **alpha** (*float, optional*) – If the transition matrix has restarts, *alpha* is the probability for the random surfer to continue (default is 0.85).

Returns

- **scores** (*list*) – The scores of all the greedy teams of size up to *k*.
- **teams** (*list*) – The list of greedy times of size up to *k*.
- **times** (*list*) – The time to compute each team. Returned only if *return_times* is True.

4.3 Matrix-related functions

compute_fundamental_matrix(P[, fast, ...])	Computes the fundamental matrix for an absorbing random walk.
compute_transition_matrix(G)	Builds the random transition matrix P.
compute_personalized_transition_matrix(G[, ...])	Returns the transition matrix of the random walk with restarts.
update_fundamental_matrix(P, F, next, previous)	Applies Woodbury's formula to update the fundamental matrix in

4.3.1 absorbing_centrality.compute_fundamental_matrix

compute_fundamental_matrix(*P*, *fast=True*, *drop_tol=1e-05*, *fill_factor=1000*)

Computes the fundamental matrix for an absorbing random walk.

Parameters

- **P** (*scipy.sparse matrix*) – The transition probability matrix of the absorbing random walk. To construct this matrix, you start from the original transition matrix and delete the rows that correspond to the absorbing nodes.
- **fast** (*bool, optional*) –
- **True (default), use the iterative SuperLU solver from** (*If*) –
- **scipy.sparse.linalg.** –
- **drop_tol** (*float, optional*) – If *fast* is True, the *drop_tol* parameter of the SuperLU solver is set to this value (default is 1e-5).
- **fill_factor** (*int, optional*) – If *If 'fast* is True, the *fill_factor* parameter of the SuperLU solver is set to this value (default is 1000).

Returns **F** – The fundamental matrix of the random walk. Element (i,j) holds the expected number of times the random walk will be in state j before absorption, when it starts from state i. For more information, check [1].

Return type scipy.sparse matrix

References

4.3.2 absorbing_centrality.compute_transition_matrix

compute_transition_matrix(*G*)

Builds the random transition matrix P. The probability of going from node *i* to node'j' is equal to:

$$P_{i,j} = \frac{1}{\text{degree}(i)}$$

Parameters **G** (*NetworkX graph*) –

[1] Doyle, Peter G., and J. Laurie Snell. Random walks and electric networks. Carus mathematical monographs 22 (2000). https://math.dartmouth.edu/~doyle/docs/walks/walks.pdf

Returns **P** – The random transition probability matrix.

Return type scipy.sparse matrix

4.3.3 absorbing_centrality.compute_personalized_transition_matrix

compute_personalized_transition_matrix(*G, alpha=0.85, restart_set=['_super_']*)

Returns the transition matrix of the random walk with restarts.

Parameters

- **G** (*graph*) –
- **alpha** (*float, optional*) – The probability of the random surfer to continue their walk (default is 0.85).
- **restart_set** (*list, optional*) – The set of nodes to restart from. If not supplied, the restarts lead to the supernode (default is [SUPER_NODE]).

Returns **P** – The probability matrix for the random walk with restarts.

Return type scipy.sparse.matrix

4.3.4 absorbing_centrality.update_fundamental_matrix

update_fundamental_matrix(*P*, *F*, *next*, *previous*, *previous_index=0*, *node_order=None*)

Applies Woodbury's formula to update the fundamental matrix in order to avoid doing an inversion.

Parameters

- **P** (*matrix*) – The transition matrix of the graph, where *previous* is non absorbing.
- **F** (*matrix*) – The fundamental matrix of the graph after setting the node *previous* as an absorbing node.
- **next** (*int*) – The node that will be set as absorbing next. The result of this call will result in a fundamental matrix where *next* is an absorbing node.
- **previous** (*int*) – The node that was set as absorbing when computing F.
- **previous_index** (*int, optional*) – The row/col index of node *previous* in P (default is 0).
- **node_order** (*list, optional*) – The nodes that corresponds to the rows/cols of *P*, in order. If not supplied, the order is considered to be [0, .. , n_P - 1], where n_P is the the number of rows/cols in *P*.

Returns

- **P_updated** (*matrix*) – The new transition matrix, where *previous* is absorbing.
- **F_updated** (*matrix*) – The fundamental matrix after adding *previous* and *next* in the set of absorbing nodes.
- **node_order_updated** (*list*) – The new order of the non absorbing nodes in the *F_new*.
- **next_index** (*int*) – The row/col index of the node *next*, that we just set as absorbing, in *P_new*.

4.4 Preprocessing the graph

canonical_relabel_nodes(G)	Relabels the nodes in the graph, such that the new names belong in the set [1,n].
is_canonical(G)	Tests if the graph has been canonicalized.
add_supernode(G[, query])	Adds a supernode to the graph and connects it with directed edges to the query nodes.
has_supernode(G)	Checks if there exist a supernode in the graph.

4.4.1 absorbing_centrality.canonical_relabel_nodes

canonical_relabel_nodes(*G*)

Relabels the nodes in the graph, such that the new names belong in the set [1,n]. The labeling information is stored in the dictionaries *G.graph['canonical_map']* and *G.graph['label_map']*. These provide a way to map original to canonical node names and vice-versa, respectively.

Parameters **G** (*NetworkX graph*) –

Returns

G_prime – The relabeled graph. It includes two attributes:

1. ***G_prime.graph['canonical_map']*** [dict] Holds the mapping between the original names of the nodes and the new, canonical, names (original -> new).
2. ***G_prime.graph['label_map']*** [dict] Holds the mapping between the new, canonical, names and the original names of the nodes (new -> original).

Return type NetworkX graph

Note: The relabeling of a particular node might not be consistent across two consecutive runs. Also, the relabeling happens on a copy, so the original graph will be untouched.

4.4.2 absorbing_centrality.is_canonical

is_canonical(*G*)

Tests if the graph has been canonicalized.

Parameters **G** (*NetworkX graph*) –

Returns

- *bool*
- *Returns True, if the graph has been canonicalized.*

4.4.3 absorbing_centrality.add_supernode

add_supernode(*G*, *query=None*)

Adds a supernode to the graph and connects it with directed edges to the query nodes.

Parameters

- **G** (*NetworkX graph*) – The graph in which we want to add a supernode.
- **query** (*list, default is None*) – The list of nodes that the supernode will be connected to. If *query* is None, the supernode will be connected to all the nodes in *G*. These new edges will be directed.

Returns A directed graph with the supernode, and the new edges, added. The attributes of the graph, i.e. 'label_map' and 'canonical_map', are also updated (or created if the input graph was not canonicalized) to reflect the new node.

Return type NetworkX graph

4.4.4 absorbing_centrality.has_supernode

has_supernode(*G*)

Checks if there exist a supernode in the graph.

Parameters **G** (*NetworkX graph*) –

Returns **has_supernode**

Return type bool

4.5 Exceptions

CanonicalizationError(message)	Exception related to the graph canonicalization procedure.

4.5.1 absorbing_centrality.CanonicalizationError

exception CanonicalizationError(*message*)

Exception related to the graph canonicalization procedure.

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

5.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.2 Documentation improvements

Absorbing Random-Walk Centrality could always use more documentation, whether as part of the official Absorbing Random-Walk Centrality docs, in docstrings, or even on the web in blog posts, articles, and such.

5.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/harrymvr/absorbing-centrality/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.4 Development

To set up *absorbing-centrality* for local development:

1. Fork absorbing-centrality on GitHub.
2. Clone your fork locally:

```
git clone git@github.com:your_name_here/absorbing-centrality.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. When you're done making changes, run all the checks, doc builder and spell checker with tox one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

5.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`) [1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

5.4.2 Tips

To run a subset of tests:

```
tox -e envname -- py.test -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

CHAPTER 6

Authors

- Charalampos Mavroforakis - http://cs-people.bu.edu/cmav
- Michael Mathioudakis - http://michalis.co
- Aristides Gionis - http://users.ics.aalto.fi/gionis/

CHAPTER 7

Changelog

7.1 0.1 (2015-08-31)

- Skeleton of the package.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

A

C

G

H

I

U