dendrogram

Release 0.3.2

Bill Chen

Apr 18, 2023

CONTENTS

CHAPTER

ONE

DENDROGRAM: N-DIMENSIONAL DENDROGRAM

`dendrogram` is a toolkit for creating N-dimensional (ND) dendrogram. See documentation for details.

1.1 Table of Contents

1.2 Intro

Placeholder

1.3 Install

The prerequisites of `dendrogram` are

```
python >= 3.8
numpy >= 1.18
```

Lower versions may also work (and higher versions may not work). Please raise an issue if it doesn't work for you. Next, you can `git clone` the source package from GitHub:

```
$ git clone https://github.com/EnthalpyBill/Dendrogram.git
```

To build and install `dendrogram`, `cd` the folder and `pip install` it:

```
$ cd Dendrogram/
$ pip install -e .
```

The `-e` command allows you to make changes to the code. Remove it if you don't want to do so. `dendrogram` has not been published to `PyPI` yet.

1.4 Usage

To use the package, import it as

```
>>> import dendrogram as dg
```

Let's consider a simple two-dimensional bimodal data.

```
>>> data = np.array([[2,1], [1,2]])
>>> print(data)
[[2 1]
 [1 2]]
```

We can easily generate the dendrogram tree with the following command:

```
>>> tree = makeTree(data, min_value=0)
```

`min_value` specifies the minimum value to consider when making the tree, and `print_progress` determines whether to print the progress or not. Other arguments include

- `min_delta` (scalar, default to 0): Lag to be ignored.
- `min_npix` (int, default to 1): Minimum number of pixels to form a cluster.
- `num_level` (int, default to 100): Number of levels.
- `print_progress` (bool, default to False): Whether to print progress or not.

The `makeTree()` method returns a `dendrogram.structureTree.clusterTree` object. To visualize it, use the following command to show the topology of the tree:

```
>>> tp = tree.topology()
└──(-1)
    └──(2)
        ├──(0)
        └──(1)
```

As expected, two branches labeled 0 and 1 illustrates the bimodality. See Module API for more details about `makeTree()`.

1.5 Contribute

Feel free to dive in! Raise an issue or submit pull requests.

1.5.1 Authors

- @mxmintaka (Xi Meng)
- @EnthalpyBill (Bill Chen)

1.5.2 Maintainers

- @EnthalpyBill (Bill Chen)

1.6 License

`dendrogram` is available at GitHub under the MIT license.

CHAPTER

TWO

MODULE API

2.1 dendrogram.lattice module

An N-dimensional lattice class with an identify_cluster method.

class dendrogram.lattice.**latticeND**(*data*, *level*)

Bases: object

An N-dimensional lattice class.

property shape

Shape of the lattice.

Type *numpy.ndarray* of int

property dim

Dimension of the lattice.

Type int

property len

Total number of elements in the lattice.

Type int

property lattice

Area ocupied by the lattice.

Type *numpy.ndarray* of int

property label

Label of clusters.

Type *numpy.ndarray* of int

identify_cluster()

Identify clusters in the lattice.

A cluster is a group of connected (neighboring) pixels.

Returns Label of clusters.

Return type *numpy.ndarray* of int

2.2 dendrogram.structureTree module

A tree class and a makeTree function.

class dendrogram.structureTree.**clusterTree**(*label*, *mask*, *isleaf=True*)
Bases: object

A tree class.

Note: *clusterTree* can be either the tree itself or the branch of a tree.

property label
Label of this tree/branch.

Type int

property mask
Area ocupied by this tree/branch.

Type *numpy.ndarray* of bool

property isleaf
Whether this is a leaf or not.

Type bool

property branches
All branches of the tree. Empty if this is a branch.

Type dict of int

property children
Children of this branch/tree. Empty if this is a leaf.

Type dict of int

property parent
Parent of this branch. Empty if this is a tree.

Type dict of int

create_leaf(*label*, *mask*)
Create a new leaf to the tree.

Note: This method does *not* connect the new leaf to any of the presenting branch!

Parameters

- **label** (*int*) – Label of the new leaf.
- **mask** (*numpy.ndarray* of bool) – The area ocupied by the new leaf.

merge_branch(*label*, *mask*, *branch*)
Merge many old branches to a new branch.

Note: Here, "merge" means setting the new branch to be the parent of old branches, *not* removing them!

Parameters

- **label** (*int*) – Label of the new branch.
- **mask** (*numpy.ndarray* of bool) – The area ocupied by the new branch.
- **branch** (*set or list of int*) – Labels of branches to be merged.

merge_final(*branch*)
Merge final branches to the tree.

Note: Even if there is only one final branch, we still need to merge (or, more properly, "link") it to the tree. Mind the difference between "tree" and "branch".

Parameters **branch** (*set or list of int*) – Labels of branches to be merged.

topology(*stdout=True*)
Print topology of the tree in pure text.

Note: Applying this method to large (with more than 100 branches) or deep (with more than 100 levels) trees is not recommend.

Parameters **stdout** (*bool*) – Whether to print to screen or not.

Returns Visualized topology of this tree.

Return type str

dendrogram.structureTree.**makeTree**(*data*, *min_value*, *min_delta=0*, *min_npix=1*, *num_level=100*, *print_progress=False*)
Make dendrogram tree from N-dimensional data.

Note: Applying this method to large (with more than 100 branches) or deep (with more than 100 levels) trees is not recommend.

Parameters

- **data** (*numpy.ndarray* of scalar) – Data to make dendrogram tree.
- **min_value** (*scalar*) – Minimum value to consider.
- **min_delta** (*scalar, default to 0*) – Lag to be ignored.
- **min_npix** (*int, default to 1*) – Minimum number of pixels to form a cluster.
- **num_level** (*int, default to 100*) – Number of levels.
- **print_progress** (*bool, default to False*) – Whether to print progress or not.

Returns Tree for the dendrogram.

Return type *clusterTree*

Examples

Consider a simple two-dimensional bimodal data, let's generate the tree with *min_value=0*:

```
>>> data = np.array([[2,1], [1,2]])
>>> tree = makeTree(data, min_value=0)
```

To check the result, we can print the topology of tree:

```
>>> tp = tree.topology()
└──(-1)
    └──(2)
        ├──(0)
        └──(1)
```

As expected, two branches 0 and 1 illustrates the bimodality. Similarly, the script below generates dendrogram for a three-peak distribution:

```
>>> data = np.array([[3,1,1],[1,1,1],[2,1,3]])
>>> tree = makeTree(data, min_value=0)
>>> tp = tree.topology()
└──(-1)
    └──(3)
        ├──(0)
        ├──(1)
        └──(2)
```

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

INDEX