Textory

Release 0.2.7b1.dev10+g0f81b8b

Jul 23, 2021

Contents

1 Installation Instructions **3**
- 1.1 Pip-based Installation . . . 3
- 1.2 Conda-based Installation . . . 3
- 1.3 Latest version . . . 3

2 Overview **5**
- 2.1 Features . . . 5

3 Basic usage **7**
- 3.1 Calculating textures . . . 7

4 Scene and Dataset wrappers **9**
- 4.1 Calculate textures for Scene . . . 9
- 4.2 Calculate textures for xarray Dataset . . . 9

5 Performance **11**

6 Textory package **13**
- 6.1 textory.textures module . . . 13
- 6.2 textory.statistics module . . . 15
- 6.3 textory.wrappers module . . . 15

7 Indices and tables **17**

Python Module Index **19**

Index **21**

Textory is a python library for the calculation of windowed statistics of arrays. It can calculate windowed variograms, pseudo-cross variograms and standard statistics like mean, std, min, max, etc. for rectangular and round windows. It can also be used conveniently with xarrays and Satpy scenes.

CHAPTER 1

Installation Instructions

1.1 Pip-based Installation

Textory can be installed from PyPi with pip:

```
$ pip install textory
```

1.2 Conda-based Installation

A pip based install from github can also be included in a conda environment.yaml file for example:

```
name: your_env_name
dependencies:
  - python=3.7
  - numpy
  - scipy
  - xarray
  - dask
  - distributed
  - pip:
    - textory
```

1.3 Latest version

Currently releases on PyPi might be infrequent. If you want to install the latest version from the github repository use:

```
$ pip install git+https://github.com/BENR0/textory.git@master#egg=textory
```

CHAPTER 2

Overview

Textory can be used to calculate correlation statistics like variogram as an image texture. Basically the statistic is calculated for a moving window over the whole array. Each pixel in the resulting array is the statistical measure of the small window around the pixel.

These type of image textures are especially interesting for machine learning (ML) approaches on spatial datasets like satellite data, since they are able to include spatial information within single bands or even between two different datasets into the ML model.

2.1 Features

Textory is able to calculate:

- Variogram: $\gamma(h) = \frac{1}{2n(h)} \sum_{i=1}^{n(h)} (v(x_i) - v(x_i + h))^2$
- Madogram: $\gamma(h) = \frac{1}{2n(h)} \sum_{i=1}^{n(h)} |v(x_i) - v(x_i + h)|$
- Rodogram: $\gamma(h) = \frac{1}{2n(h)} \sum_{i=1}^{n(h)} \sqrt{|v(x_i) - v(x_i + h)|}$
- Cross Variogram: $\gamma(h) = \frac{1}{2n(h)} \sum_{i=1}^{n(h)} (v(x_i) - v(x_i + h)) * (w(x_i) - w(x_i + h))$
- Pseudo Cross Variogram: $\gamma(h) = \frac{1}{2n(h)} \sum_{i=1}^{n(h)} (v(x_i) - w(x_i + h))^2$
- basic statistics (e.g. min, max, median, etc. (only for square windows))
- TPI (Topographic Position Index) for different window sizes and geometries.

for different lag distances and window sizes (round and square windows) for numpy and `dask.array.Array` as well as `xarray.DataArray`. Furthermore convenient functions to easily caluculate these statistics for `xarray.Dataset` and `satpy.scene.Scene` are available.

CHAPTER 3

Basic usage

3.1 Calculating textures

All available textures are located in the `textures` submodule and except for the windowed basic statistics *`window_statistic()`*, have the same parameters.

First we import necessary packages and create some dummy data:

```
import textory as tx
import numpy as np

n = 50
data1 = np.random.rand(n*n).reshape(n,n)
data2 = np.random.rand(n*n).reshape(n,n)
```

Then we can calculate, for example the variogram, with:

```
tx.textures.variogram(x=data1, lag=2, win_size=7)
```

Here the parameter *win_geom* is ommitted and therefore defaults to "square". The `lag` and `win_size` parameters can also be omitted in which case they default to *1* and *5* respectively.

Textures based on two images like *`pseudo_cross_variogram()`* have an additional parameter `y`:

```
tx.textures.pseudo_cross_variogram(x=data1, y=data2)
```

With the *`window_statistic()`* function basic statistical measures like min, max median, etc. can be calculated for a moving window. To get accurate results the nan version of the numpy functions should be used. Currently this function only supports square windows.

```
tx.textures.window_statistic(x=data1, stat="nanmax")
```

CHAPTER 4

Scene and Dataset wrappers

Textory satpy Scene and xarray Dataset wrappers

4.1 Calculate textures for Scene

With the *textures_for_scene()* function it is easy to calculate one or multiple textures with the same or different parameters for datasets in a satpy.scene.Scene. The function will return a new Scene either with the textures in addition to all input datasets (default) or a Scene only with the textures, depending on the append parameter of the function.

The textures parameter takes a dictionary where the keys are a tuple with the texture and the parameters which to calculate and the values are a list (or list of tuples in the case of textures which require two inputs) of the datasets to apply the texture to in the general form of:

```
textures_dict = {("texture_name", lag, win_size, win_geom): [list of dataset names]}
```

The following example would calculate the variogram with lag=2, win_size=7, win_geom="square" for the datasets with name "IR_039" and "IR_108" as well as the cross variogram with lag=1, win_size=5, and win_geom="round" between the datasets with name "WV_062" and "IR_108" of the Scene.

```
import textory as tx

scn = Scene(...)
textures_dict = {("variogram", 2, 7, "square"): ["IR_039", "IR_108"],
                 ("cross_variogram", 1, 5, "round"): [("WV_062", "IR_108")]}
scn_with_textures = tx.textures_for_scene(scn, textures=textures_dict)
```

4.2 Calculate textures for xarray Dataset

The *textures_for_xr_dataset()* function works similarly to the *textures_for_scene()* function above but takes xarray.Dataset as input and also returns a xarray.Dataset.

CHAPTER 5

Performance

In general performance of Textory shold be good. For very small arrays, even though performance should be quite similiar, it is slightly faster if the textures are used with numpy arrays instead of dask arrays since they have a small overhead (see Dask documentation for details about that).

One thing to note for is; for very large windows, even though textory can use dask, memory might be a problem. Of course the exact window size limit depends on the actuall amount of memory available in the system used. For example for a system with 16Gb of memory window sizes below ~190 should be possible.

The reason for this problem is the usage of `convolve()` which is know to have inefficient memory management for large window sizes.

Some hints for calculating with large windows:

- choose smaller chunk sizes than normaly would be efficent
- reduce the number of chunks processed at the same time. For a local Dask cluster limit the number of threads for example

```
import dask
from multiprocessing.pool import ThreadPool

dask.config.set(pool=ThreadPool(1))
```

Of course limiting the number of chunks processed at the same time mitigates the performance increase normally seen while using Dask.

CHAPTER 6

Textory package

6.1 textory.textures module

textory.textures.**cross_variogram**(*x*, *y*, *lag=1*, *win_size=5*, *win_geom='square'*, ***kwargs*)
Calculate moveing window pseudo-variogram with specified lag for the two arrays.

Parameters

- **y** (*x,*) – Input array
- **lag** (*int*) – Lag distance for variogram, defaults to 1.
- **win_size** (*int, optional*) – Length of one side of window. Window will be of size window*window.
- **win_geom** (*{"square", "round"}*) – Geometry of the kernel. Defaults to square.

Returns *array like* – Array where each element is the pseudo-variogram between the two arrays of the window around the element.

textory.textures.**madogram**(*x*, *lag=1*, *win_size=5*, *win_geom='square'*, ***kwargs*)
Calculate moveing window madogram with specified lag for array.

Parameters

- **x** (*array like*) – Input array
- **lag** (*int*) – Lag distance for variogram, defaults to 1.
- **win_size** (*int, optional*) – Length of one side of window. Window will be of size window*window.
- **win_geom** (*{"square", "round"}*) – Geometry of the kernel. Defaults to square.

Returns *array like* – Array where each element is the madogram of the window around the element

textory.textures.**pseudo_cross_variogram**(*x*, *y*, *lag=1*, *win_size=5*, *win_geom='square'*, ***kwargs*)
Calculate moveing window pseudo-variogram with specified lag for the two arrays.

Parameters

- **y** (*x,*) – Input array
- **lag** (*int*) – Lag distance for variogram, defaults to 1.
- **win_size** (*int, optional*) – Length of one side of window. Window will be of size window*window.
- **win_geom** (*{"square", "round"}*) – Geometry of the kernel. Defaults to square.

Returns *array like* – Array where each element is the pseudo-variogram between the two arrays of the window around the element.

textory.textures.**rodogram**(*x*, *lag=1*, *win_size=5*, *win_geom='square'*, ***kwargs*)
Calculate moveing window rodogram with specified lag for array.

Parameters

- **x** (*array like*) – Input array
- **lag** (*int*) – Lag distance for variogram, defaults to 1.
- **win_size** (*int, optional*) – Length of one side of window. Window will be of size window*window.
- **win_geom** (*{"square", "round"}*) – Geometry of the kernel. Defaults to square.

Returns *array like* – Array where each element is the madogram of the window around the element

textory.textures.**tpi**(*x*, *win_size=5*, *win_geom='square'*, ***kwargs*)
Calculate topographic position index for a given window size.

Parameters

- **x** (*array like*) – Input array
- **win_size** (*int, optional*) – Length of one side of window. Window will be of size window*window. Defaults to 5.
- **win_geom** (*{"square", "round"}*) – Geometry of the kernel. Defaults to square.

Returns *array like* – Array with tpi

textory.textures.**variogram**(*x*, *lag=1*, *win_size=5*, *win_geom='square'*, ***kwargs*)
Calculate moveing window variogram with specified lag for array.

Parameters

- **x** (*array like*) – Input array
- **lag** (*int*) – Lag distance for variogram, defaults to 1.
- **win_size** (*int, optional*) – Length of one side of window. Window will be of size window*window.
- **win_geom** (*{"square", "round"}*) – Geometry of the kernel. Defaults to square.

Returns *array like* – Array where each element is the variogram of the window around the element

textory.textures.**window_statistic**(*x*, *stat='nanmean'*, *win_size=5*, ***kwargs*)
Calculate the specified statistic with a moveing window of size *win_size*.

Parameters

- **x** (*array like*) – Input array

- **stat** (*{"nanmean", "nanmax", "nanmin", "nanmedian", "nanstd"}*) – Statistical measure to calculate.
- **win_size** (*int, optional*) – Length of one side of window. Window will be of size window*window.
- **kwargs** (*optional*) – Any parameters a certain stat may need other than the array itself.

Returns *array like*

6.2 textory.statistics module

textory.statistics.**pseudo_cross_variogram**(*x*, *y*, *lag=1*)

Calculate pseudo-variogram with specified lag for the two arrays.

Parameters

- **y** (*x,*) – Input arrays
- **lag** (*int*) – Lag distance for variogram, defaults to 1.

Returns *float* – Pseudo-variogram between the two arrays

textory.statistics.**variogram**(*x*, *lag=1*)

Calculate variogram with specified lag for array.

Parameters

- **x** (*array like*) – Input array
- **lag** (*int*) – Lag distance for variogram, defaults to 1.

Returns *float* – Variogram

6.3 textory.wrappers module

Textory satpy Scene and xarray Dataset wrappers

6.3.1 Calculate textures for Scene

With the *textures_for_scene()* function it is easy to calculate one or multiple textures with the same or different parameters for datasets in a satpy.scene.Scene. The function will return a new Scene either with the textures in addition to all input datasets (default) or a Scene only with the textures, depending on the append parameter of the function.

The textures parameter takes a dictionary where the keys are a tuple with the texture and the parameters which to calculate and the values are a list (or list of tuples in the case of textures which require two inputs) of the datasets to apply the texture to in the general form of:

```
textures_dict = {("texture_name", lag, win_size, win_geom): [list of dataset names]}
```

The following example would calculate the variogram with lag=2, win_size=7, win_geom="square" for the datasets with name "IR_039" and "IR_108" as well as the cross variogram with lag=1, win_size=5, and win_geom="round" between the datasets with name "WV_062" and "IR_108" of the Scene.

```
import textory as tx

scn = Scene(...)
textures_dict = {("variogram", 2, 7, "square"): ["IR_039", "IR_108"],
                 ("cross_variogram", 1, 5, "round"): [("WV_062", "IR_108")]}
scn_with_textures = tx.textures_for_scene(scn, textures=textures_dict)
```

6.3.2 Calculate textures for xarray Dataset

The *textures_for_xr_dataset()* function works similarly to the *textures_for_scene()* function above but takes xarray.Dataset as input and also returns a xarray.Dataset.

textory.wrappers.**textures_for_scene**(*scn*, *textures*, *append=True*)

Wrapper to calculate multiple textures for datasets in a satpy.scene.Scene.

Parameters

- **scn** (*satpy.scene.Scene*) –
- **textures** (*dict*) – Dictionary with textures bands to calulate. The accepted notation is {("texture name", lag, win_size, win_geom): [list of dataset names or tuples of dataset names]}

  For example: {("variogram", 2, 7, "round"), ["IR_039", "IR_103"]}
- **append** (*boolean, optional*) – If *False* returns a new satpy.scene.Scene with all calculated textures, By default returns a new satpy.scene.Scene with all input datasets and all calculated textures.

Returns *satpy.scene.Scene*

textory.wrappers.**textures_for_xr_dataset**(*xrds*, *textures*, *append=True*)

Wrapper to calculate multiple textures for dataarrays in a xarray.Dataset.

Parameters

- **xrds** (*xarray.Dataset*) –
- **textures** (*dict*) – Dictionary with textures bands to calulate. The accepted notation is {("texture name", lag, win_size, win_geom): [list of dataset names or tuples of dataset names]}

  For example: {("variogram", 2, 7, "round"), ["IR_039", "IR_103"]}
- **append** (*boolean, optional*) – If *False* returns a new xarray.Dataset with all calculated textures, By default returns a new xarray.Dataset with all input datasets and all calculated textures.

Returns *xarray.Dataset* – Dataset

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

textory.statistics, 15
textory.textures, 13
textory.wrappers, 15

Index

C

cross_variogram() (*in module textory.textures*), 13

M

madogram() (*in module textory.textures*), 13

P

pseudo_cross_variogram() (*in module textory.statistics*), 15
pseudo_cross_variogram() (*in module textory.textures*), 13

R

rodogram() (*in module textory.textures*), 14

T

textory.statistics (*module*), 15
textory.textures (*module*), 13
textory.wrappers (*module*), 15
textures_for_scene() (*in module textory.wrappers*), 16
textures_for_xr_dataset() (*in module textory.wrappers*), 16
tpi() (*in module textory.textures*), 14

V

variogram() (*in module textory.statistics*), 15
variogram() (*in module textory.textures*), 14

W

window_statistic() (*in module textory.textures*), 14