nd Documentation

Johannes Hansen

Nov 01, 2021

INTRODUCTION

1 Changelog **3**
- 1.1 Version 0.4 *(under development)* . . . 3
- 1.2 Version 0.3.1 . . . 3
- 1.3 Version 0.3 . . . 3
- 1.4 Version 0.2 . . . 4

2 User Guide **5**
- 2.1 Installing nd . . . 5
- 2.2 Getting Started . . . 5
- 2.3 Reading and writing datasets . . . 6
- 2.4 Using nd with `xarray` . . . 7
- 2.5 Projections . . . 7
- 2.6 Data Visualization . . . 11
- 2.7 Classification . . . 13
- 2.8 Change Detection . . . 17
- 2.9 Filters . . . 18

3 Reference **21**
- 3.1 nd.change package . . . 21
- 3.2 nd.classify package . . . 22
- 3.3 nd.filters package . . . 23
- 3.4 nd.io package . . . 28
- 3.5 nd.testing package . . . 30
- 3.6 nd.tiling package . . . 30
- 3.7 nd.utils package . . . 32
- 3.8 nd.vector package . . . 35
- 3.9 nd.visualize package . . . 36
- 3.10 nd.warp package . . . 37
- 3.11 `xarray` accessors . . . 43

Python Module Index **49**

Index **51**

This package contains a selection of tools to handle and analyze satellite data. nd is making heavy use of the xarray and rasterio libraries. The GDAL library is only used via rasterio as a compatibility layer in nd.io to enable reading supported file formats. Internally, all data is passed around as xarray Datasets and all provided methods expect this format as inputs. *nd.io.open_rasterio()* may be used to convert any GDAL-readable file into an xarray.Dataset.

An xarray.Dataset is essentially a Python representation of the NetCDF file format and as such easily reads/writes NetCDF files.

What does this library add?

xarray provides all data structures required for dealing with *n*-dimensional data in Python. nd explicitly does not aim to add additional data structures or file formats. Rather, the aim is to bring the various corners of the scientific ecosystem in Python closer together.

As such, nd adds functionality to more seamlessly integrate libraries like xarray, rasterio, scikit-learn, etc.

For example:

- nd allows to reproject an entire multivariate and multi-temporal dataset between different coordinate systems by wrapping rasterio methods.
- nd provides a wrapper for scikit-learn estimators to easily apply classification algorithms to raster data.

Additionally, nd contains a growing library of algorithms that are especially useful for spatio-temporal datacubes, for example:

- change detection algorithms
- spatio-temporal filters

Since xarray is our library of choice for representing geospatial raster data, this is also an attempt to promote the use of xarray and the NetCDF file format in the Earth Observation community.

Why NetCDF?

NetCDF (specifically NetCDF-4) is a highly efficient file format that was built on top of HDF5. It is capable of random access which ties in with indexing and slicing in numpy. Because slices of a large dataset can be accessed independently, it becomes feasible to handle larger-than-memory file sizes. NetCDF-4 also supports data compression using zlib. Random access capability for compressed data is maintained through data chunking. Furthermore, NetCDF is designed to be fully self-descriptive. Crucially, it has a concept of named dimensions and coordinates, can store units and arbitrary metadata.

CHAPTER

ONE

CHANGELOG

1.1 Version 0.4 *(under development)*

1.1.1 General updates

- …

1.2 Version 0.3.1

1.2.1 `nd.classify`

- added *nd.classify.Classifier.make_Xy()*
- added *nd.classify.Classifier.score()*

1.2.2 `nd.warp`

- small compatibility updates for new versions of shapely

1.3 Version 0.3

1.3.1 General updates

- drop support for Python 3.5
- `cartopy` is now an optional dependency

1.3.2 nd.utils

- use `multiprocessing` rather than `dask.delayed` in *nd.utils.parallel()*

1.4 Version 0.2

1.4.1 General updates

- add support for Python 3.8
- make `libgsl` dependency optional

1.4.2 nd.classify

- removed redundant method `nd.classify.cluster()`, as same functionality can be achieved using *nd.classify.Classifier*

1.4.3 nd.tiling

- added *nd.tiling.debuffer()* to automatically remove buffer from tiled datasets

1.4.4 nd.utils

- added *nd.utils.apply()* to apply functions with specified signature to arbitrary subsets of dataset dimensions

1.4.5 nd.visualize

- added *nd.visualize.plot_map()* to plot the geometry of a dataset on a map
- added `nd.visualize.gridlines_with_labels()` to add perfectly aligned tick labels around a map with gridlines

CHAPTER

TWO

USER GUIDE

2.1 Installing nd

You may also want to install the GDAL library, but `rasterio` comes with a stripped down version of GDAL so for most use cases this should not be necessary.

The easiest way to install `nd` is via `pip` from PyPI:

```
pip install numpy
pip install nd
```

You can also install the latest version from Github:

```
pip install git+https://github.com/jnhansen/nd
```

Some algorithms require the `libgsl-dev` C library:

- `nd.change.OmnibusTest`

If you want to use these algorithms you need to make sure you have the library installed *before* installing `nd`. You can find out whether it is installed by checking if the command `gsl-config` exists on your machine.

2.1.1 Rebuilding the C extensions from Cython

In case you want to rebuild the C extensions from the `.pyx` files, you need to install the additional dependencies `cython` and `cythongsl`. With those installed, `pip install` will automatically regenerate the C files prior to compilation.

2.2 Getting Started

Read a file:

```
from nd.io import open_dataset
ds = open_dataset('data.tif')
```

For details on how to work magic with xarray Datasets, refer to the xarray documentation.

2.3 Reading and writing datasets

As nd is built around xarray, most of the IO is handled by xarray. However, nd provides an extra layer of abstraction, specifically with the functions nd.open_dataset() and nd.to_netcdf().

2.3.1 Reading a dataset

Ideally, your data exists already in the netCDF file format. However, as most geospatial data is distributed as GeoTiff or other file formats, nd and xarray rely on rasterio as a Python-friendly wrapper around GDAL for dealing with such raster data.

The *nd.io* module contains three additional functions to read different file formats:

- *nd.io.open_netcdf()* to read a NetCDF file
- *nd.io.open_rasterio()* to read a rasterio/GDAL readable file
- *nd.io.open_beam_dimap()* to read the BEAM Dimap format, which is the best supported format in SNAP.

as well as the convenience function nd.open_dataset() which resorts to one of the three functions above based on the file extension. All of these return xarray.Dataset or xarray.DataArray objects.

Most of the algorithms work on both Dataset and DataArray objects.

2.3.2 Writing a dataset

Write your processed data to disk using nd.to_netcdf().

Note: Currently, it is assumed that you will only ever want to convert your data from other formats into netCDF, but not the other way around. So if you need to export your result as a GeoTiff, you are on your own (for now). Sorry about that!

Here is a list of things that nd.open_dataset and nd.to_netcdf do in addition to xarray.open_dataset and xarray.Dataset.to_netcdf:

- Handle complex-valued data. NetCDF doesn't support complex valued data, so before writing to disk, complex variables are disassembled into their real and imaginary parts. After reading from disk, these parts can be reassembled into complex valued variables. That means if you use the functions provided by nd you don't have to worry about complex-valued data at all.
- Provide x and y coordinate arrays even if the NetCDF file uses lat and lon nomenclature. This is to be consistent with the general case of arbitrary projections.

```
>>> import nd
>>> import xarray as xr
>>> path = 'data/C2.nc'
>>> ds_nd = nd.open_dataset(path)
>>> ds_xr = xr.open_dataset(path)
>>> {v: ds_nd[v].dtype for v in ds_nd.data_vars}
{'C11': dtype('<f4'),
 'C22': dtype('<f4'),
 'C12': dtype('complex64')}
>>> {v: ds_xr[v].dtype for v in ds_xr.data_vars}
{'C11': dtype('float32'),
```

(continues on next page)

(continued from previous page)

```
 'C12__im': dtype('float32'),
 'C12__re': dtype('float32'),
 'C22': dtype('float32')}
```

See Also:

- http://xarray.pydata.org/en/stable/io.html

2.4 Using nd with xarray

nd is built on top of and around xarray. As such, it is meant to be used with xarray.Dataset and xarray.DataArray objects.

Much of the functionality contained in nd is available directly from these xarray objects via custom accessors after importing the library.

2.4.1 Applying filters using the filter accessor

```
import xarray as xr
import nd
ds = xr.open_dataset('data/C2.nc')
ds_filtered = ds.filter.gaussian(dims=('y', 'x'), sigma=0.5)
```

2.4.2 Reprojecting a dataset using the nd accessor

```
import xarray as xr
import nd
ds = xr.open_dataset('data/C2.nc')
ds_proj = ds.nd.reproject(crs='EPSG:27700')
```

2.5 Projections

nd handles geographic projections with rasterio. Projection information is usually stored in the metadata attributes crs and transform, but different standards exist.

All functionality related to coordinate systems and projections is contained in the module *nd.warp*. You can extract the coordinate reference system of a dataset using *nd.warp.get_crs()*:

```
>>> from nd.warp import get_crs
>>> get_crs(ds)
CRS.from_epsg(3086)
```

The returned object is always of type rasterio.crs.CRS.

Similarly, the coordinate transformation can be extracted using *nd.warp.get_transform()*:

```
>>> from nd.warp import get_transform
>>> get_transform(ds)
Affine(178.27973915722524, 0.0, 572867.3883891336,
       0.0, -178.27973915722524, 622453.9641249835)
```

which is an `affine.Affine` object and represents the mapping from image coordinates to projection coordinates.

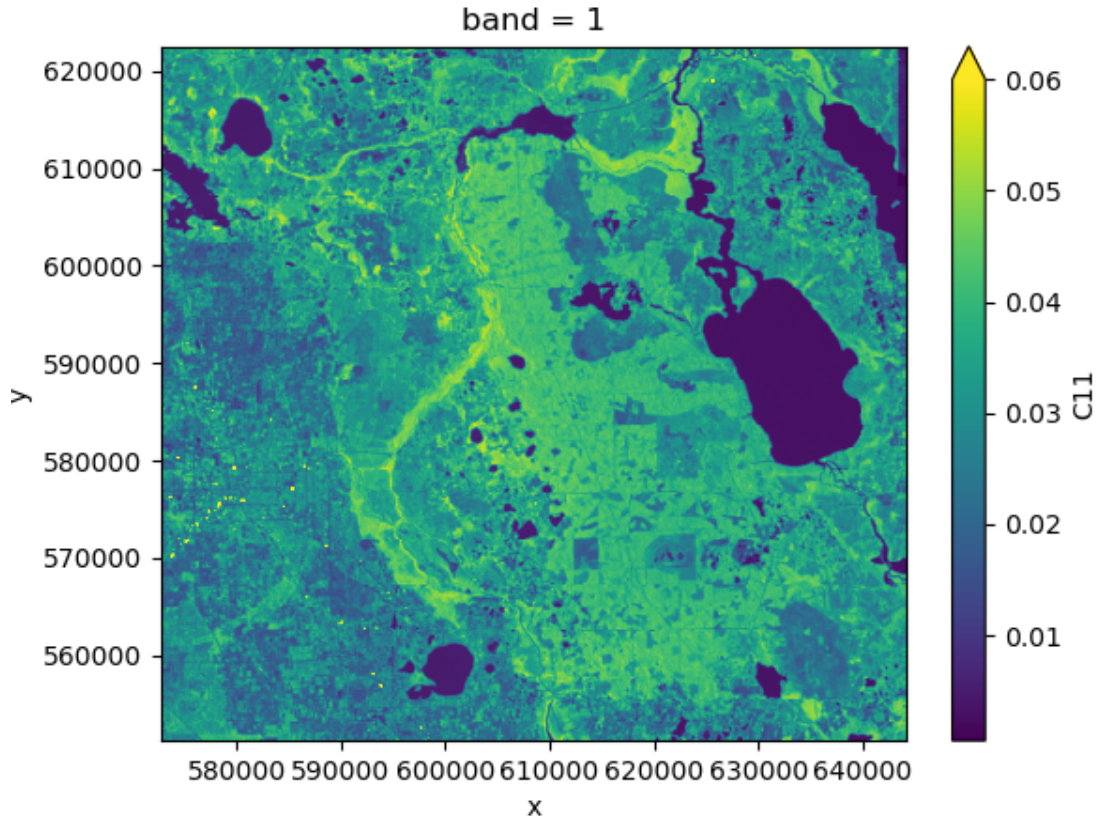
Additionally, a dataset contains the coordinates of the x and y axes in `ds.coords['x']` and `ds.coords['y']`. The transform object and the coordinate arrays represent the same information.

2.5.1 Reprojecting to a different CRS

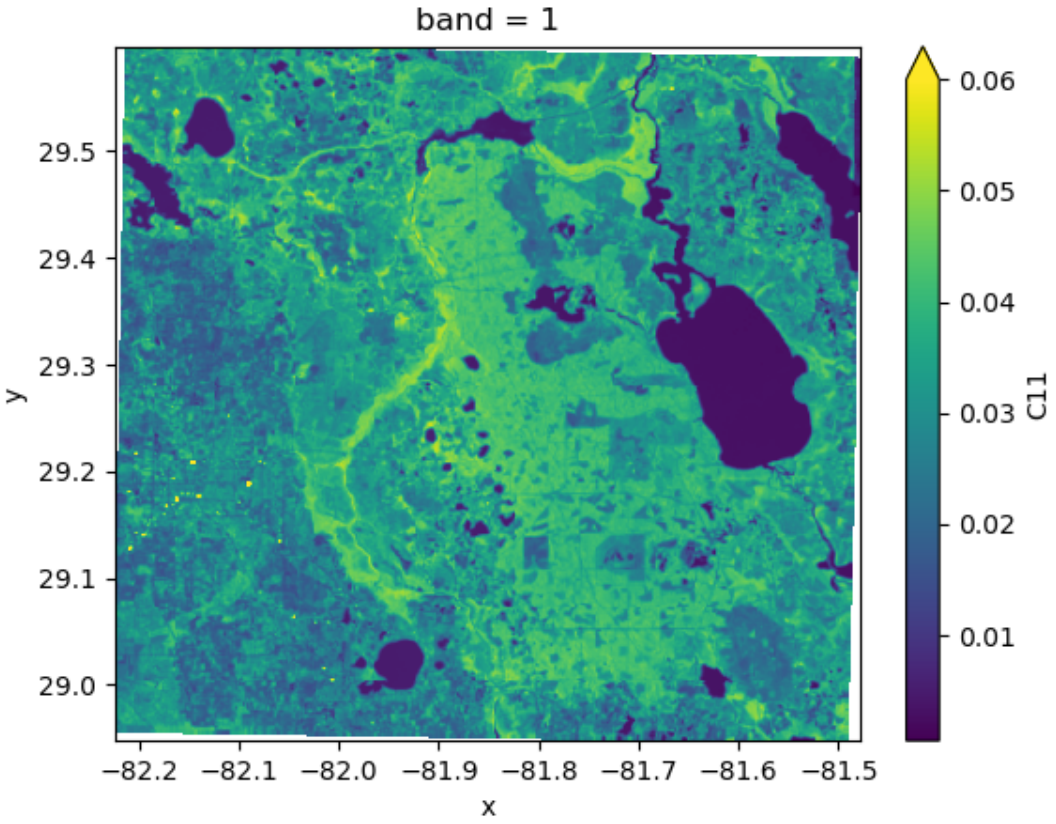
You can reproject your dataset to a different coordinate system using *nd.warp.Reprojection*. For example, the following code will reproject your dataset into Web Mercator (EPSG:3857):

```
>>> from nd.warp import Reprojection, get_crs
>>> get_crs(ds)
CRS.from_epsg(3086)
>>> proj = Reprojection(crs='EPSG:3857')
>>> ds_reprojected = proj.apply(ds)
>>> get_crs(ds_reprojected)
CRS.from_epsg(3857)
```

```
>>> from nd.io import open_dataset
>>> ds = open_dataset('data/C2.nc')
>>> ds.C11.mean('time').plot(vmax=0.06)
```




```
>>> from nd.warp import Reprojection
>>> epsg4326 = Reprojection(crs='epsg:4326')
>>> proj = epsg4326.apply(ds)
>>> proj.C11.mean('time').plot(vmax=0.06)
```



`Reprojection()` lets you specify many more options, such as the desired extent and resolution.

When reprojecting a dataset this way, `nd` will also add coordinate arrays `lat` and `lon` to the result which contains the latitude and longitude values at a number of tie points, irrespective of the projection. Storing these arrays alongside the projection information allows GIS software to correctly display the data.

See Also:

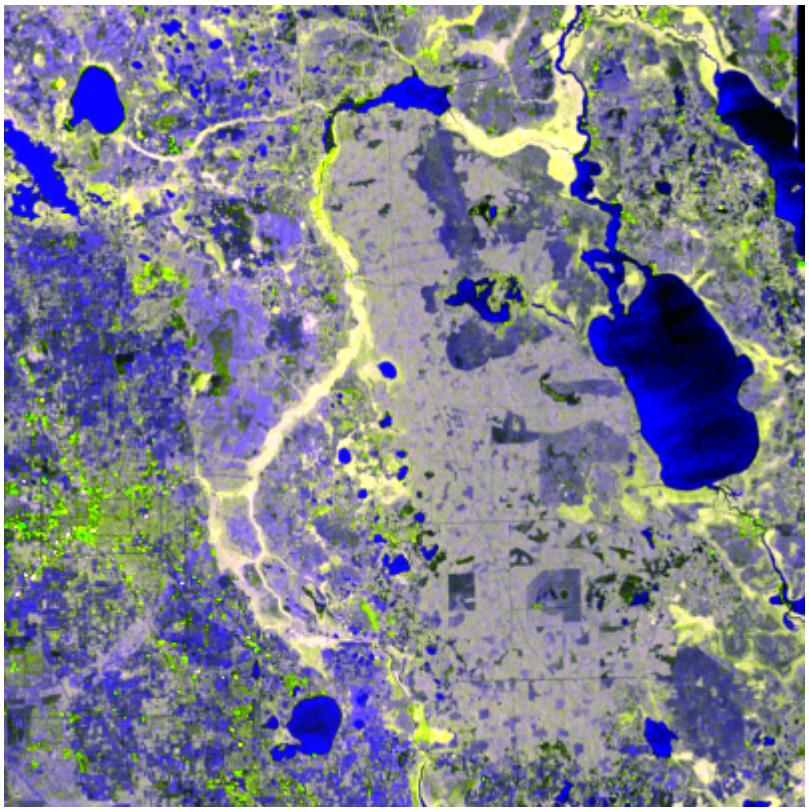
- *nd.warp*
- https://rasterio.readthedocs.io/en/latest/topics/georeferencing.html
- https://rasterio.readthedocs.io/en/latest/topics/reproject.html

2.6 Data Visualization

When you want to plot a single variable for a single time slice, `xarray`'s built-in plotting methods are more than sufficient. `nd` provides a few extra methods for visualizing multi-temporal and multivariate datasets.

2.6.1 Creating RGB composites from multivariate data

```
>>> from nd.io import open_dataset
>>> from nd.visualize import to_rgb
>>> ds = open_dataset('data/C2.nc')
>>> t0 = ds.isel(time=0)
>>> rgb = to_rgb([t0.C11, t0.C22, t0.C11 / t0.C22], 'images/c2_rgb.png')
```



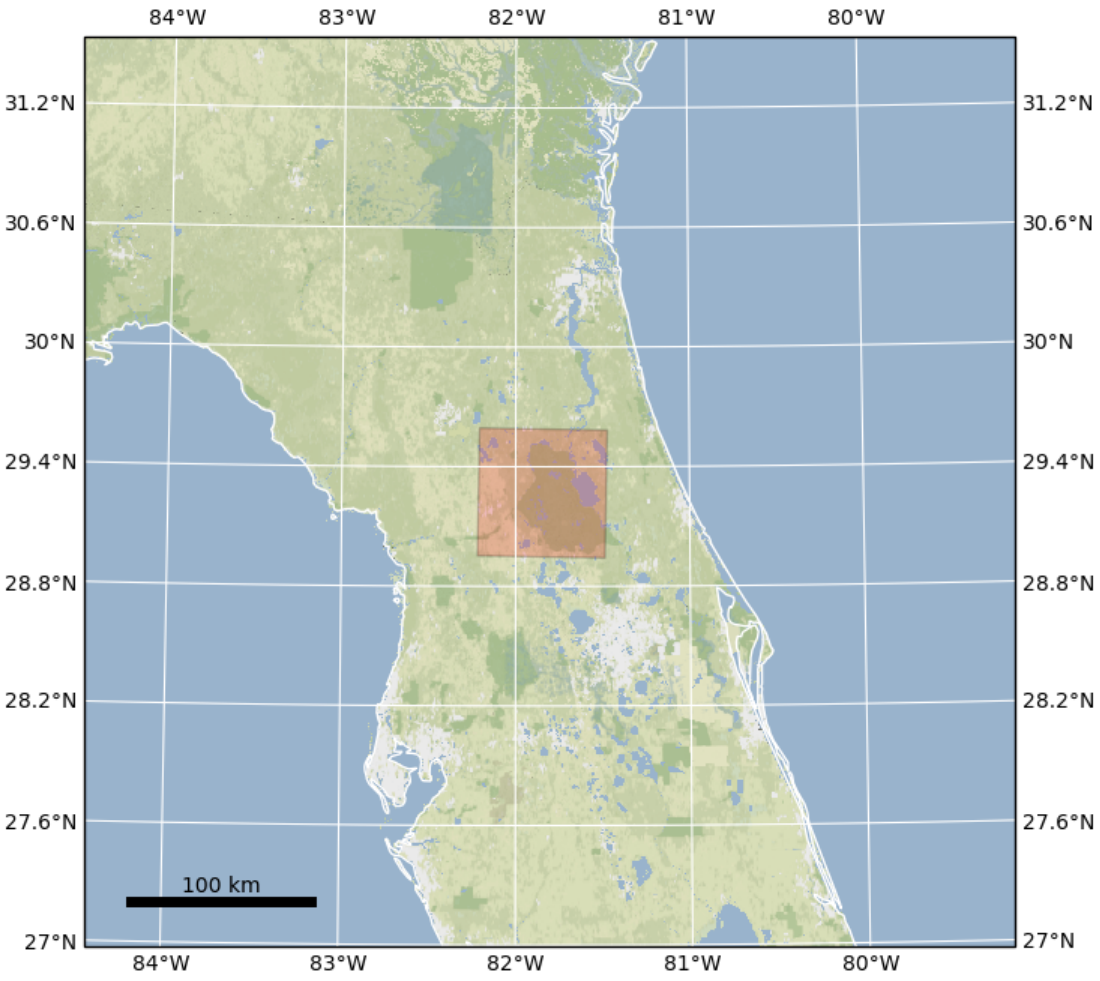
2.6.2 Creating video from time series data

The example writes to a gif image so it can be embedded here, but many video formats are supported.

```
>>> from nd.visualize import write_video
>>> write_video(ds, 'images/c2.gif', fps=5, timestamp=False)
```

2.6.3 Creating a map

```
>>> from nd.visualize import plot_map
>>> import matplotlib.pyplot as plt
>>> plt.figure()
>>> plot_map(ds, buffer=6, imscale=11)
```



See Also:

- http://xarray.pydata.org/en/stable/plotting.html

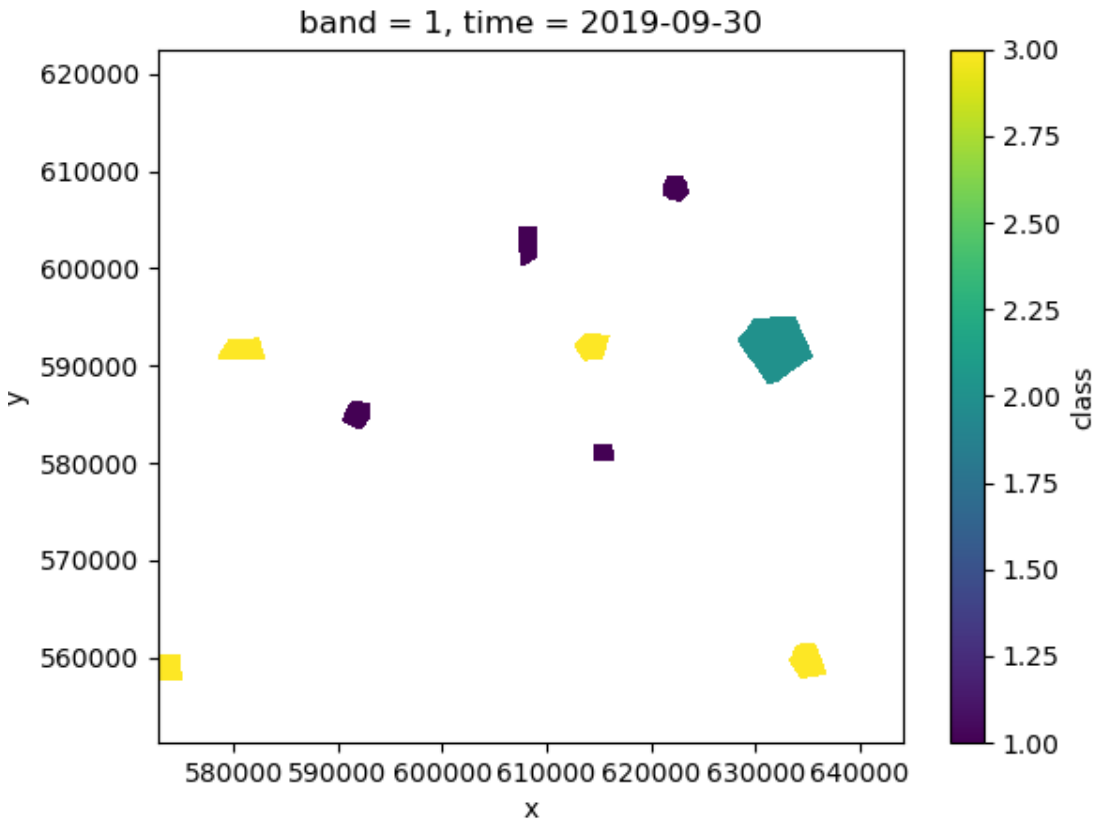
2.7 Classification

nd provides a wrapper for `scikit-learn` classifiers that can be directly trained on and applied to `xarray` Datasets.

2.7.1 Training a classifier

First, we need to get some training data. We will do a forest/non-forest classification using some polygon training data, which we can rasterize to match the dataset:

```
>>> import nd
>>> from nd.classify import Classifier
>>> from sklearn.ensemble import RandomForestClassifier
>>> path = '../data/C2.nc'
>>> ds = nd.open_dataset(path)
>>> labels = nd.vector.rasterize('../data/labels.shp', ds)['class'].squeeze()
>>> labels.where(labels > 0).plot()
```



If we investigate `labels`, we see that it has an associated legend to match the integer classes:

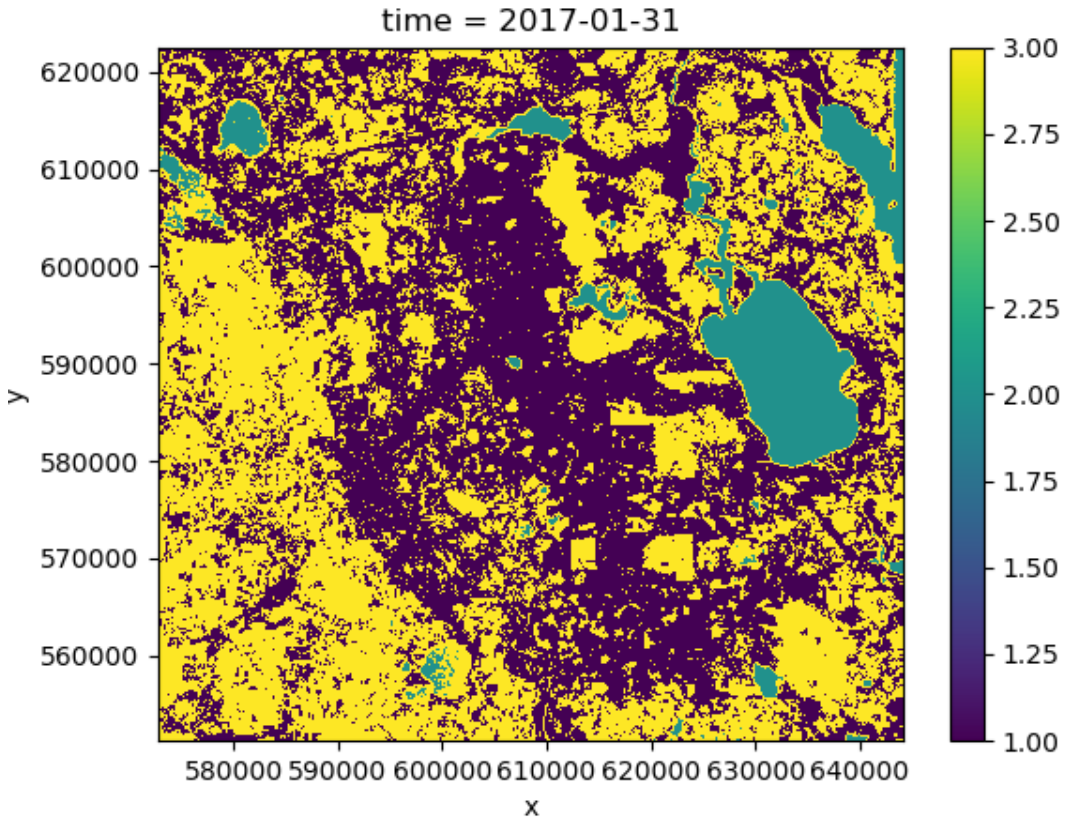
```
>>> labels
<xarray.DataArray 'class' (y: 400, x: 400)>
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
```

(continues on next page)

(continued from previous page)

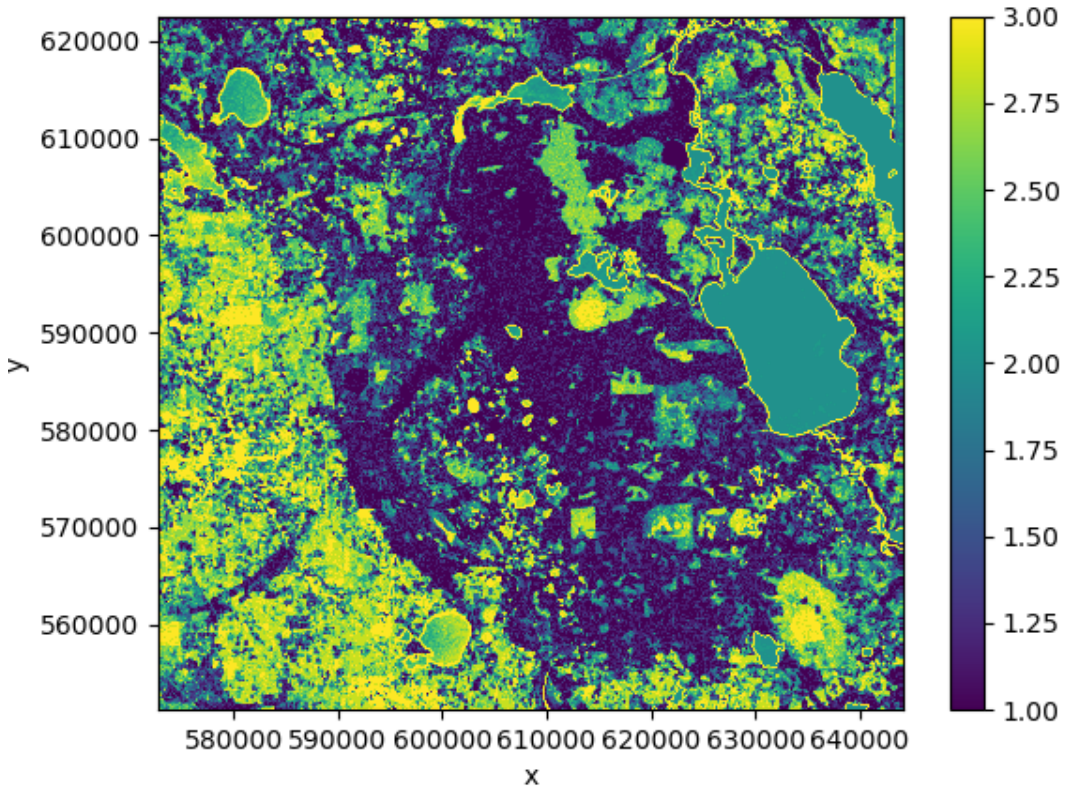
```
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]])
Coordinates:
    band     int64 1
  * y        (y) float64 6.225e+05 6.223e+05 6.221e+05 ... 5.515e+05 5.513e+05
  * x        (x) float64 5.729e+05 5.73e+05 5.732e+05 ... 6.438e+05 6.44e+05
    time     datetime64[ns] 2019-09-30
Attributes:
    legend:   [(0, None), (1, 'forest'), (2, 'water'), (3, 'nonforest')]
```

```
>>> clf = Classifier(RandomForestClassifier(n_estimators=10))
>>> pred = clf.fit(ds, labels).predict(ds)
>>> pred.isel(time=0).plot()
```



If we plot the mean of the predicted class over time we can see that the predictions change because the forest cover changes over the course of the time period:

```
>>> pred.mean('time').plot()
```

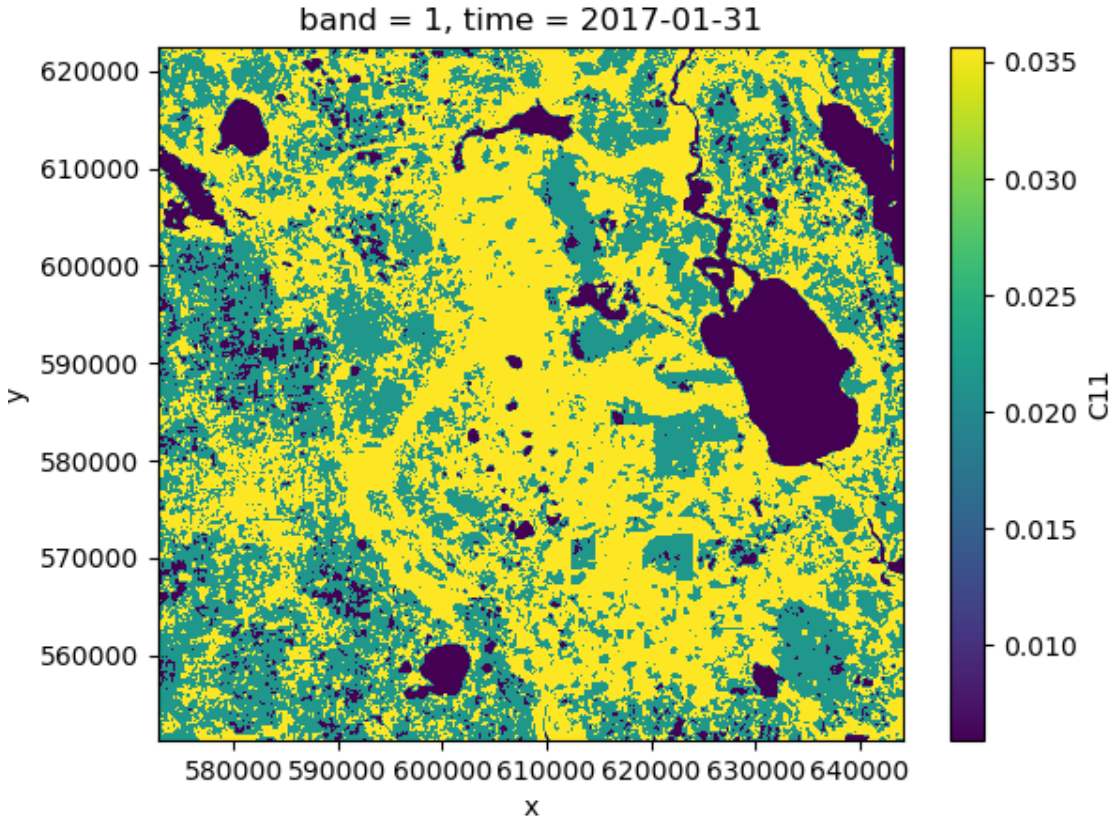


2.7.2 Clustering

Clustering can be done using the same `Classifier` object because clustering classes in `scikit-learn` provide the same interface as classifiers. Clustering is an unsupervised approach, so the `labels` argument will be omitted.

In the following example, we are using *nd.classify.class_mean()* to replace every pixel with the mean of its cluster for visualization:

```
>>> from sklearn.cluster import MiniBatchKMeans
>>> from nd.classify import class_mean
>>> clf = Classifier(MiniBatchKMeans(n_clusters=3))
>>> pred = clf.fit_predict(ds.isel(time=0))
>>> means = class_mean(ds.isel(time=0), pred)
>>> means.C11.plot()
```



It is advisable to use a clustering algorithm that scales well, such as `MiniBatchKMeans`. Alternatively, one can fit the clusterer to a smaller subset of the data by applying a mask.

2.7.3 Feature and data dimensions

Internally, the entire dataset needs to be converted to a two-dimensional array to work with most classification algorithm in `scikit-learn`.

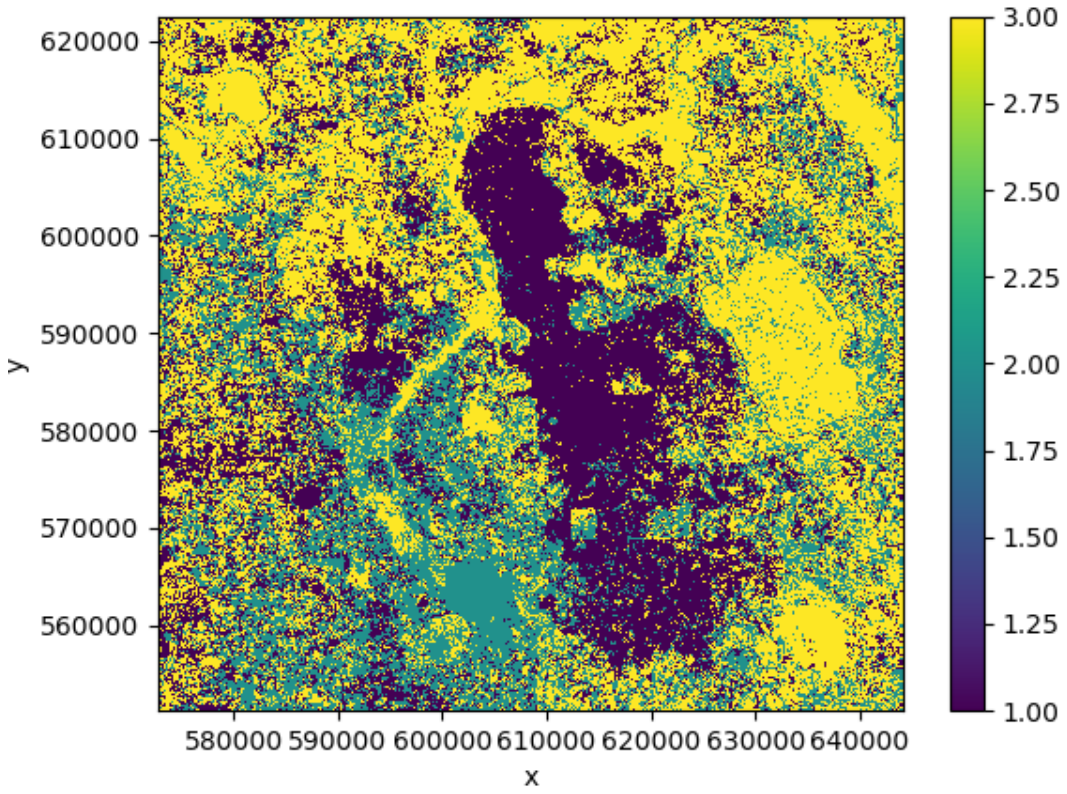
The first dimension (rows) corresponds to independent data points, whereas the second dimension (columns) corresponds to the features (attributes, variables) of that data point.

By default, `nd` will flatten all dataset dimensions into the rows of the array, and convert the data variables into the columns of the array.

However, *`nd.classify.Classifier`* has an additional keyword argument `feature_dims` that controls which dimensions are considered to be features of the data. Typically, this could be a `band` dimension, which really isn't a dimensions but a set of features (or variables) of the data. It could also be the `time` dimension, in which case all time steps are treated as additional information about a point, rather than separate points in the feature space.

Example:

```
>>> clf = Classifier(RandomForestClassifier(n_estimators=10),
...                  feature_dims=['time'])
>>> pred = clf.fit(ds, labels).predict(ds)
>>> pred.plot()
```



Our prediction output no longer has a `time` dimension because it was converted into a feature dimension and used for prediction. In this case the result is not great because the classes change over time and we thus have noisy training data.

See Also:

- *nd.classify*
- https://scikit-learn.org/

2.8 Change Detection

2.8.1 Omnibus Test Change Detection

Conradsen et al. (2016) present a change detection algorithm for time series of complex valued SAR data based on the complex Wishart distribution for the covariance matrices. S_{rt} denotes the complex scattering amplitude where $r, t \in \{h, v\}$ are the receive and transmit polarization, respectively (horizontal or vertical). Reciprocity is assumed, i.e. $S_{hv} = S_{vh}$. Then the backscatter at a single pixel is fully represented by the complex target vector

$$\boldsymbol{s} = \begin{bmatrix} S_{hh} & S_{hv} & S_{vv} \end{bmatrix}^T$$

For multi-looked SAR data, backscatter values are averaged over n pixels (to reduce speckle) and the backscatter may be represented appropriately by the (variance-)covariance matrix, which for fully polarimetric SAR data is given by

$$\langle C \rangle_{\text{full}} = \langle \boldsymbol{s}(i)\boldsymbol{s}(i)^H \rangle = \begin{bmatrix} \langle S_{hh}S_{hh}^* \rangle & \langle S_{hh}S_{hv}^* \rangle & \langle S_{hh}S_{vv}^* \rangle \\ \langle S_{hv}S_{hh}^* \rangle & \langle S_{hv}S_{hv}^* \rangle & \langle S_{hv}S_{vv}^* \rangle \\ \langle S_{vv}S_{hh}^* \rangle & \langle S_{vv}S_{hv}^* \rangle & \langle S_{vv}S_{vv}^* \rangle \end{bmatrix}$$

where $\langle \cdot \rangle$ is the ensemble average, * denotes complex conjugation, and ^{H} is Hermitian conjugation. Often, only one polarization is transmitted (e.g. horizontal), giving rise to dual polarimetric SAR data. In this case the covariance matrix is

$$\langle C \rangle_{\text{dual}} = \begin{bmatrix} \langle S_{hh}S_{hh}^* \rangle & \langle S_{hh}S_{hv}^* \rangle \\ \langle S_{hv}S_{hh}^* \rangle & \langle S_{hv}S_{hv}^* \rangle \end{bmatrix}$$

These covariance matrices follow a complex Wishart distribution as follows:

$$\boldsymbol{X}_i \sim W_C(p, n, \boldsymbol{\Sigma}_i), \quad i = 1, ..., k$$

where p is the rank of $\boldsymbol{X}_i = n\langle \boldsymbol{C}_i \rangle$, $E[\boldsymbol{X}_i] = n\boldsymbol{\Sigma}_i$, and $\boldsymbol{\Sigma}_i$ is the expected value of the covariance matrix.

In the first instance, the change detection problem then becomes a test of the null hypothesis $H_0 : \boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}_2 = ... = \boldsymbol{\Sigma}_k$, i.e. whether the expected value of the backscatter remains constant. This test is a so-called omnibus test.

A test statistic for the omnibus test can be derived as:

$$Q = k^{pnk} \frac{\prod_{i=1}^{k} |\boldsymbol{X}_i|^n}{|\boldsymbol{X}|^{nk}} = \left\{ k^{pk} \frac{\prod_{i=1}^{k} |\boldsymbol{X}_i|}{|\boldsymbol{X}|^k} \right\}^n$$

where $\boldsymbol{X} = \sum_{i=1}^{k} \boldsymbol{X}_i \sim W_C(p, nk, \boldsymbol{\Sigma})$. The test statistic can be translated into a probability $p(H_0)$. The hypothesis test is repeated iteratively over subsets of the time series in order to determine the actual time of change.

See Also:

- *nd.change.OmnibusTest*

References:

- Conradsen, K., Nielsen, A. A., & Skriver, H. (2016). Determining the Points of Change in Time Series of Polarimetric SAR Data. IEEE Transactions on Geoscience and Remote Sensing, 54(5), 3007–3024.

2.9 Filters

A filter is a transform that assigns to every pixel a weighted average of its surrounding pixels. Different filters differ in how the weights are computed. Because all filters are conceptually very similar, they are grouped together in the submodule *nd.filters*.

2.9.1 Kernel Convolutions

nd.filters implements generic kernel convolutions as well as several convenience functions for special cases. Every filter supports the argument `dims` to specify a subset of dimensions along which to apply the filter. If a kernel is given, the number of dimensions must match the shape of the kernel.

The following example performs a simple Sobel edge detection filter along the `x` dimension using *nd.filters.ConvolutionFilter()*.

Example:

```
from nd.filters import ConvolutionFilter
kernel = np.array([[1, 0, -1],
                   [2, 0, -2],
                   [1, 0, -1]])
edges = ConvolutionFilter(kernel, dims=('y', 'x'))
edges = conv.apply(ds)
```

A boxcar convolution (see *nd.filters.BoxcarFilter()*) uses a square kernel (in *n* dimensions) with equal weights for each pixel. The total weight is normalized to one.

Example:

```
from nd.filters import BoxcarFilter
boxcar = BoxcarFilter(w=3, dims=('y', 'x'))
smooth = boxcar.apply(ds)
```

A Gaussian filter (see *nd.filters.GaussianFilter()*) convolves the datacube with a Gaussian kernel.

Example:

```
from nd.filters import GaussianFilter
gaussian = GaussianFilter(sigma=1, dims=('y', 'x'))
smooth = gaussian.apply(ds)
```

2.9.2 Non-Local Means

Non-Local Means is a denoising filter that computes filtered pixel values as a weighted average of pixels in the spatial neighborhood, where the weights are determined as a function of color distance.

Example:

```
from nd.filters import NLMeansFilter
nlm = NLMeansFilter(dims=('y', 'x', 'time'), r=(3, 3, 1),
                    sigma=1, h=1, f=1)
ds_filtered = nlm.apply(ds)
```

See Also:

- *nd.filters.NLMeansFilter()*

References:

- Buades, A., Coll, B., & Morel, J.-M. (2011). Non-Local Means Denoising. Image Processing On Line, 1, 208–212.

CHAPTER

THREE

REFERENCE

3.1 nd.change package

class nd.change.**ChangeDetection**(*njobs=1*)
Bases: nd.algorithm.Algorithm

njobs = 1

class nd.change.**OmnibusTest**(*ml=None*, *n=1*, *alpha=0.01*, **args*, ***kwargs*)
Bases: *nd.change.ChangeDetection*

This class implements the change detection algorithm by Conradsen et al. (2015).

Parameters

- **ds** (*xarray.Dataset*) – A (multilooked) dataset in covariance matrix format.
- **ml** (*int, optional*) – Multilooking window size. By default, no multilooking is performed and the dataset is assumed to already be multilooked.
- **n** (*int, optional*) – The number of looks in *ds*. If *ml* is specified this parameter is ignored (default: 1).
- **alpha** (*float (0. ... 1.), optional*) – The significance level (default: 0.01).
- **kwargs** (*dict, optional*) – Extra keyword arguments to be applied to ChangeDetection.__init__.

apply(*ds*)
Must be implemented by derived classes and should be given @parallelize decorator where appropriate.

nd.change.**omnibus**(*ds*, *ml=None*, *n=1*, *alpha=0.01*, **args*, ***kwargs*)
Wrapper for *nd.change.OmnibusTest*.

OmnibusTest

This class implements the change detection algorithm by Conradsen et al. (2015).

Parameters

- **ds** (*xarray.Dataset*) – A (multilooked) dataset in covariance matrix format.
- **ml** (*int, optional*) – Multilooking window size. By default, no multilooking is performed and the dataset is assumed to already be multilooked.
- **n** (*int, optional*) – The number of looks in *ds*. If *ml* is specified this parameter is ignored (default: 1).
- **alpha** (*float (0. ... 1.), optional*) – The significance level (default: 0.01).

- **kwargs** (*dict, optional*) – Extra keyword arguments to be applied to `ChangeDetection.__init__`.

3.2 nd.classify package

class nd.classify.**Classifier**(*clf*, *feature_dims=[]*, *scale=False*)

Bases: `object`

Parameters

- **clf** (*sklearn classifier*) – An initialized classifier object as provided by `scikit-learn`. Must provide methods `fit` and `predict`.
- **feature_dims** (*list, optional*) – A list of additional dimensions to use as features. For example, if the dataset has a `'time'` dimension and `'time'` is in `feature_dims`, every time step will be treated as an independent variable for classification purposes. Otherwise, all time steps will be treated as additional data dimensions just like `'x'` and `'y'`.
- **scale** (*bool, optional*) – If True, scale the input data before clustering to zero mean and unit variance (default: False).

fit(*ds*, *labels=None*)

Parameters

- **ds** (*xarray.Dataset*) – The dataset on which to train the classifier.
- **labels** (*xarray.DataArray, optional*) – The class labels to train the classifier. To be omitted if the classifier is unsupervised, such as KMeans.

fit_predict(*ds*, *labels=None*)

make_Xy(*ds*, *labels=None*)

Generate scikit-learn compatible X and y arrays from *ds* and *labels*.

Parameters

- **ds** (*xarray.Dataset*) – The input dataset.
- **labels** (*xarray.DataArray*) – The corresponding class labels.

Returns X and y

Return type tuple(np.array, np.array)

predict(*ds*, *func='predict'*)

Parameters

- **ds** (*xarray.Dataset*) – The dataset for which to predict the class labels.
- **func** (*str, optional*) – The method of the classifier to use for prediction (default: `'predict'`).

Returns The predicted class labels.

Return type xarray.DataArray

score(*ds*, *labels=None*, *method='accuracy'*)

Compute the classification score.

Parameters

- **ds** (*xarray.Dataset*) – The dataset for which to compute the score.
- **labels** (*xarray.DataArray*) – The corresponding true class labels.
- **method** (*str, optional*) – The scoring method as implemented in scikit-learn (default: 'accuracy')

Returns The classification score.

Return type float

nd.classify.**class_mean**(*ds*, *labels*)

Replace every pixel of the dataset with the mean of its corresponding class (or cluster, or segment).

Parameters

- **ds** (*xarray.Dataset*) – The dataset.
- **labels** (*xarray.DataArray*) – The labels indicating the class each pixel in `ds` belongs to. The label dimensions may be a subset of the dimensions of `ds` (such as `('y', 'x')` for a dataset that also contains a `time` dimension but with time-independent class labels).

Returns A dataset with the corresponding mean class values.

Return type xarray.Dataset

3.3 nd.filters package

The main use of image filters is for noise reduction. This module implements several such filters, all of which are designed to work in an arbitrary number of dimensions.

nd.filters.**_expand_kernel**(*kernel*, *kernel_dims*, *new_dims*)

Reshape a kernel spanning some dimensions to cover a superset of dimensions.

Parameters

- **kernel** (*ndarray*) – An n-dimensional kernel.
- **kernel_dims** (*tuple*) – The dimensions corresponding to the kernel axes.
- **new_dims** (*tuple*) – The dimensions of the dataset to which the kernel needs to be applied. Must be a superset of *kernel_dims*.

Returns The reshaped kernel.

Return type ndarray

Raises

- **ValueError** – Will raise a ValueError if the dimensions of the kernel don't match *kernel_dims*.
- **ValueError** – Will raise a ValueError if *new_dims* is not a superset of *kernel_dims*.

class nd.filters.**BoxcarFilter**(*dims=('y', 'x')*, *w=3*, ***kwargs*)

Bases: *nd.filters.ConvolutionFilter*

A boxcar filter.

Parameters

- **dims** (*tuple of str, optional*) – The dimensions along which to apply the filter (default: ('y', 'x')).

- **w** (*int*) – The width of the boxcar window. Should be an odd integer in order to ensure symmetry.
- **kwargs** (*dict, optional*) – Extra keyword arguments passed on to `scipy.ndimage.filters.convolve`.

apply(*ds*, *inplace=False*, ***, *njobs=1*)
Apply the filter to the input dataset.

Parameters

- **ds** (*xarray.Dataset*) – The input dataset
- **inplace** (*bool, optional*) – If True, overwrite the input data inplace (default: False).
- **njobs** (*int, optional*) – Number of jobs to run in parallel. Setting njobs to -1 uses the number of available cores. Disable parallelism by setting njobs to 1 (default).

Returns The filtered dataset

Return type xarray.Dataset

dims = ()

kwargs = {}

per_variable = True

supports_complex = True

class nd.filters.**ConvolutionFilter**(*dims=('y', 'x')*, *kernel=None*, ***kwargs*)
Bases: *nd.filters.Filter*

Kernel-convolution of an xarray.Dataset.

Parameters

- **dims** (*tuple, optional*) – The dataset dimensions corresponding to the kernel axes (default: ('y', 'x')). The length of the tuple must match the number of dimensions of the kernel.
- **kernel** (*ndarray*) – The convolution kernel.
- **kwargs** (*dict, optional*) – Extra keyword arguments passed on to `scipy.ndimage.filters.convolve`.

apply(*ds*, *inplace=False*, ***, *njobs=1*)
Apply the filter to the input dataset.

Parameters

- **ds** (*xarray.Dataset*) – The input dataset
- **inplace** (*bool, optional*) – If True, overwrite the input data inplace (default: False).
- **njobs** (*int, optional*) – Number of jobs to run in parallel. Setting njobs to -1 uses the number of available cores. Disable parallelism by setting njobs to 1 (default).

Returns The filtered dataset

Return type xarray.Dataset

dims = ()

kwargs = {}

per_variable = True

supports_complex = True

class nd.filters.**Filter**(**args*, ***kwargs*)
Bases: nd.algorithm.Algorithm

The base class for a generic filter.

Parameters **dims** (*tuple of str*) – The dimensions along which the filter is applied.

apply(*ds*, *inplace=False*, ***, *njobs=1*)
Apply the filter to the input dataset.

Parameters

- **ds** (*xarray.Dataset*) – The input dataset
- **inplace** (*bool, optional*) – If True, overwrite the input data inplace (default: False).
- **njobs** (*int, optional*) – Number of jobs to run in parallel. Setting njobs to -1 uses the number of available cores. Disable parallelism by setting njobs to 1 (default).

Returns The filtered dataset

Return type xarray.Dataset

dims = ()

per_variable = True

supports_complex = False

class nd.filters.**GaussianFilter**(*dims=('y', 'x')*, *sigma=1*, ***kwargs*)
Bases: *nd.filters.Filter*

A Gaussian filter.

Parameters

- **dims** (*tuple of str, optional*) – The dimensions along which to apply the Gaussian filtering (default: ('y', 'x')).
- **sigma** (*float or sequence of float*) – The standard deviation for the Gaussian kernel. If sequence, this is set individually for each dimension.
- **kwargs** (*dict, optional*) – Extra keyword arguments passed on to scipy.ndimage.filters.gaussian_filter.

Returns The filtered dataset.

Return type xarray.Dataset

apply(*ds*, *inplace=False*, ***, *njobs=1*)
Apply the filter to the input dataset.

Parameters

- **ds** (*xarray.Dataset*) – The input dataset
- **inplace** (*bool, optional*) – If True, overwrite the input data inplace (default: False).
- **njobs** (*int, optional*) – Number of jobs to run in parallel. Setting njobs to -1 uses the number of available cores. Disable parallelism by setting njobs to 1 (default).

Returns The filtered dataset

Return type xarray.Dataset

dims = ()

per_variable = True

supports_complex = False

class nd.filters.**NLMeansFilter**(*dims=('y', 'x')*, *r=1*, *sigma=1*, *h=1*, *f=1*, *n_eff=- 1*)
Bases: *nd.filters.Filter*

Non-Local Means (Buades2011).

Buades, A., Coll, B., & Morel, J.-M. (2011). Non-Local Means Denoising. Image Processing On Line, 1, 208–212. https://doi.org/10.5201/ipol.2011.bcm_nlm

Parameters

- **dims** (*tuple of str*) – The dataset dimensions along which to filter.
- **r** (*{int, sequence}*) – The radius
- **sigma** (*float*) – The standard deviation of the noise present in the data.
- **h** (*float*) –
- **f** (*int*) –
- **n_eff** (*float, optional*) – The desired effective sample size. If given, must be greater than 1 and should be no larger than about half the pixels in the window. -1 means no fixed effective sample size (default: -1).

apply(*ds*, *inplace=False*, ***, *njobs=1*)
Apply the filter to the input dataset.

Parameters

- **ds** (*xarray.Dataset*) – The input dataset
- **inplace** (*bool, optional*) – If True, overwrite the input data inplace (default: False).
- **njobs** (*int, optional*) – Number of jobs to run in parallel. Setting njobs to -1 uses the number of available cores. Disable parallelism by setting njobs to 1 (default).

Returns The filtered dataset

Return type xarray.Dataset

dims = ()

per_variable = False

supports_complex = False

nd.filters.**boxcar**(*ds*, *inplace=False*, *dims=('y', 'x')*, *w=3*, ***, *njobs=1*, ***kwargs*)
Wrapper for *nd.filters.BoxcarFilter*.

A boxcar filter.

Parameters

- **ds** (*xarray.Dataset*) – The input dataset
- **inplace** (*bool, optional*) – If True, overwrite the input data inplace (default: False).
- **dims** (*tuple of str, optional*) – The dimensions along which to apply the filter (default: ('y', 'x')).
- **w** (*int*) – The width of the boxcar window. Should be an odd integer in order to ensure symmetry.
- **njobs** (*int, optional*) – Number of jobs to run in parallel. Setting njobs to -1 uses the number of available cores. Disable parallelism by setting njobs to 1 (default).

- **kwargs** (*dict, optional*) – Extra keyword arguments passed on to `scipy.ndimage.filters.convolve`.

Returns The filtered dataset

Return type xarray.Dataset

nd.filters.**convolution**(*ds*, *inplace=False*, *dims=('y', 'x')*, *kernel=None*, ***, *njobs=1*, ***kwargs*)
Wrapper for *nd.filters.ConvolutionFilter*.

Kernel-convolution of an xarray.Dataset.

Parameters

- **ds** (*xarray.Dataset*) – The input dataset
- **inplace** (*bool, optional*) – If True, overwrite the input data inplace (default: False).
- **dims** (*tuple, optional*) – The dataset dimensions corresponding to the kernel axes (default: ('y', 'x')). The length of the tuple must match the number of dimensions of the kernel.
- **kernel** (*ndarray*) – The convolution kernel.
- **njobs** (*int, optional*) – Number of jobs to run in parallel. Setting njobs to -1 uses the number of available cores. Disable parallelism by setting njobs to 1 (default).
- **kwargs** (*dict, optional*) – Extra keyword arguments passed on to `scipy.ndimage.filters.convolve`.

Returns The filtered dataset

Return type xarray.Dataset

nd.filters.**gaussian**(*ds*, *inplace=False*, *dims=('y', 'x')*, *sigma=1*, ***, *njobs=1*, ***kwargs*)
Wrapper for *nd.filters.GaussianFilter*.

A Gaussian filter.

Parameters

- **ds** (*xarray.Dataset*) – The input dataset
- **inplace** (*bool, optional*) – If True, overwrite the input data inplace (default: False).
- **dims** (*tuple of str, optional*) – The dimensions along which to apply the Gaussian filtering (default: ('y', 'x')).
- **sigma** (*float or sequence of float*) – The standard deviation for the Gaussian kernel. If sequence, this is set individually for each dimension.
- **njobs** (*int, optional*) – Number of jobs to run in parallel. Setting njobs to -1 uses the number of available cores. Disable parallelism by setting njobs to 1 (default).
- **kwargs** (*dict, optional*) – Extra keyword arguments passed on to `scipy.ndimage.filters.gaussian_filter`.

Returns The filtered dataset

Return type xarray.Dataset

nd.filters.**nlmeans**(*ds*, *inplace=False*, *dims=('y', 'x')*, *r=1*, *sigma=1*, *h=1*, *f=1*, *n_eff=- 1*, ***, *njobs=1*)
Wrapper for *nd.filters.NLMeansFilter*.

Non-Local Means (Buades2011).

Buades, A., Coll, B., & Morel, J.-M. (2011). Non-Local Means Denoising. Image Processing On Line, 1, 208–212. https://doi.org/10.5201/ipol.2011.bcm_nlm

Parameters

- **ds** (*xarray.Dataset*) – The input dataset
- **inplace** (*bool, optional*) – If True, overwrite the input data inplace (default: False).
- **dims** (*tuple of str*) – The dataset dimensions along which to filter.
- **r** (*{int, sequence}*) – The radius
- **sigma** (*float*) – The standard deviation of the noise present in the data.
- **h** (*float*) –
- **f** (*int*) –
- **n_eff** (*float, optional*) – The desired effective sample size. If given, must be greater than 1 and should be no larger than about half the pixels in the window. -1 means no fixed effective sample size (default: -1).
- **njobs** (*int, optional*) – Number of jobs to run in parallel. Setting njobs to -1 uses the number of available cores. Disable parallelism by setting njobs to 1 (default).

Returns The filtered dataset

Return type xarray.Dataset

3.4 nd.io package

nd.io.**add_time**(*ds*, *inplace=False*)
Add a *time* dimension to the dataset.

Parameters **ds** (*xarray.Dataset*) – The input dataset.

Returns A dataset that is guaranteed to contain the dimension *time*.

Return type xarray.Dataset

nd.io.**assemble_complex**(*ds*, *inplace=False*)
Reassemble complex valued data.

NOTE: Changes the dataset (view) in place!

Parameters

- **ds** (*xarray.Dataset*) – The input dataset with complex variables split into real and imaginary parts.
- **inplace** (*bool, optional*) – Whether to modify the dataset inplace (default: False).

Returns If inplace, returns None. Otherwise, returns a dataset where the real and imaginary parts have been combined into the respective complex variables.

Return type xarray.Dataset or None

nd.io.**disassemble_complex**(*ds*, *inplace=False*)
Disassemble complex valued data into real and imag parts.

Parameters

- **ds** (*xarray.Dataset*) – The input dataset with complex variables.
- **inplace** (*bool, optional*) – Whether to modify the dataset inplace (default: False).

Returns If inplace, returns None. Otherwise, returns a dataset where all complex variables have been split into their real and imaginary parts.

Return type xarray.Dataset or None

nd.io.**open_beam_dimap**(*path*, *read_data=True*, *as_complex=True*)
Read a BEAM Dimap product into an xarray Dataset.

BEAM Dimap is the native file format of the SNAP software. It consists of a `*.dim` XML file and a `*.data` directory containing the data. `path` should point to the XML file.

Parameters

- **path** (*str*) – The file path to the BEAM Dimap product.
- **read_data** (*bool, optional*) – If True (default), read all data. Otherwise, read only the metadata.

Returns The same dataset converted into xarray.

Return type xarray.Dataset

nd.io.**open_dataset**(*path*, **args*, ***kwargs*)
The default way of reading a dataset from disk.

Determines the file format from the extension, and calls either *nd.io.open_netcdf()*, *nd.io.open_beam_dimap()*, or *nd.io.open_rasterio()*.

Parameters

- **path** (*str*) – The file path.
- ***args** (*list*) – Extra positional arguments passed on to the specialized `open_*` function.
- ****kwargs** (*dict*) – Extra keyword arguments passed on to the specialized `open_*` function.

Returns The opened dataset. In general, if the file is a NetCDF or BEAM-Dimap file the result will be an xarray Dataset, otherwise an xarray DataArray.

Return type xarray.Dataset or xarray.DataArray

Raises **IOError** – Raises an IOError if the Dataset could not be opened.

nd.io.**open_netcdf**(*path*, *as_complex=False*, **args*, ***kwargs*)
Read a NetCDF file into an xarray Dataset.

Wrapper function for `xarray.open_dataset` that preserves complex valued data.

Parameters

- **path** (*str*) – The path of the NetCDF file to read.
- **as_complex** (*bool, optional*) – Whether or not to assemble real and imaginary parts into complex (default: False).
- ***args** (*list*) – Extra positional arguments passed on to `xarray.open_dataset`.
- ****kwargs** (*dict*) – Extra keyword arguments passed on to `xarray.open_dataset`.

Returns The opened dataset.

Return type xarray.Dataset

See also:

- `xarray.open_dataset`

nd.io.**open_rasterio**(*path*, **args*, ***kwargs*)

nd.io.**to_netcdf**(*ds*, *path*, **args*, ***kwargs*)

Write an xarray Dataset to disk.

In addition to `xarray.to_netcdf`, this function allows to store complex valued data by converting it to a a pair of reals. This process is reverted when reading the file via `from_netcdf`.

Parameters

- **ds** (*xarray.Dataset*) – The dataset to be stored to disk.
- **path** (*str*) – The path of the target NetCDF file.
- ***args** (*list*) – Extra positional arguments for `xr.Dataset.to_netcdf`.
- ****kwargs** (*dict*) – Extra keyword arguments for `xr.Dataset.to_netcdf`.

3.5 nd.testing package

3.6 nd.tiling package

This module may be used to mosaic and tile multiple satellite image products.

TODO: Contain buffer information in NetCDF metadata?

nd.tiling.**auto_merge**(*datasets*, *buffer=True*, *chunks={}*, *meta_variables=[]*, *use_xarray_combine=True*)

Automatically merge a split xarray Dataset. This is designed to behave like `xarray.open_mfdataset`, except it supports concatenation along multiple dimensions.

Parameters

- **datasets** (*str or list of str or list of xarray.Dataset*) – Either a glob expression or list of paths as you would pass to xarray.open_mfdataset, or a list of xarray datasets. If a list of datasets is passed, you should make sure that they are represented as dask arrays to avoid reading the whole dataset into memory.
- **buffer** (*bool, optional*) – If True, attempt to automatically remove any buffer from the tiles (default: True).
- **meta_variables** (*list, optional*) – A list of metadata items to concatenate as variables.
- **use_xarray_combine** (*bool, optional*) – Use xr.combine_by_coords to combine the datasets (default: True). Only available from `xarray>=0.12.2`. Will fallback to a custom implementation if `False` or unavailable.

Returns The merged dataset.

Return type xarray.Dataset

nd.tiling.**debuffer**(*datasets*, *flat=True*)

Remove buffer from tiled datasets.

Parameters

- **datasets** (*list of xr.Dataset*) – The overlapping tiles.
- **flat** (*bool, optional*) – If `True`, return a flat list. Otherwise, return a numpy array representing the correct order of the tiles (default: True).

nd.tiling.**map_over_tiles**(*files*, *fn*, *args=()*, *kwargs={}*, *path=None*, *suffix=''*, *merge=True*, *overwrite=False*, *compute=True*)

Apply function to each tile.

Parameters

- **files** (*str or list of str*) – A glob expression matching all tiles. May also be a list of file paths.
- **fn** (*function*) – The function to apply to each tile.
- **args** (*tuple, optional*) – Additional arguments to fn
- **kwargs** (*dict, optional*) – Additional keyword arguments to fn
- **path** (*str, optional*) – The output directory. If None, write to same directory as input files. (default: None)
- **suffix** (*str, optional*) – If input file is *part.0.nc*, will create *part.0{suffix}.nc*. (default: '')
- **merge** (*bool, optional*) – If True, return a merged view of the result (default: True).
- **overwrite** (*bool, optional*) – Force overwriting existing files (default: False).
- **compute** (*bool, optional*) – If True, compute result immediately. Otherwise, return a dask.delayed object (default: True).

Returns The (merged) output dataset.

Return type xarray.Dataset

nd.tiling.**sort_into_array**(*datasets*, *dims=None*)

Create an array corresponding to the way the datasets are tiled.

nd.tiling.**sort_key**(*ds*, *dims*)

To be used as key when sorting datasets.

nd.tiling.**tile**(*ds*, *path*, *prefix='part'*, *chunks=None*, *buffer=0*)

Split dataset into tiles and write to disk. If *chunks* is not given, use chunks in dataset.

Parameters

- **ds** (*xarray.Dataset*) – The dataset to split into tiles.
- **path** (*str*) – The output directory in which to place the tiles.
- **prefix** (*str, optional*) – The tile names will start with `{prefix}.`
- **chunks** (*dict, optional*) – A dictionary of the chunksize for every dimension along which to split. By default, use the dask array chunksize if the data is represented as dask arrays.
- **buffer** (*int or dict, optional*) – The number of overlapping pixels to store around each tile (default: 0). Can be given as an integer or per dimension as dictionary.

3.7 nd.utils package

This module provides several helper functions.

nd.utils.**apply**(*ds*, *fn*, *signature=None*, *njobs=1*)
Apply a function to a Dataset that operates on a defined subset of dimensions.

Parameters

- **ds** (*xr.Dataset or xr.DataArray*) – The dataset to which to apply the function.
- **fn** (*function*) – The function to apply to the Dataset.
- **signature** (*str, optional*) – The signature of the function in dimension names, e.g. '(time,var)->(time)'. If 'var' is included, the Dataset variables will be converted into a new dimension and the result will be a DataArray.
- **njobs** (*int, optional*) – The number of jobs to run in parallel.

Returns The output dataset with changed dimensions according to the function signature.

Return type xr.Dataset

nd.utils.**array_chunks**(*array*, *n*, *axis=0*, *return_indices=False*)
Chunk an array along the given axis.

Parameters

- **array** (*numpy.array*) – The array to be chunked
- **n** (*int*) – The chunksize.
- **axis** (*int, optional*) – The axis along which to split the array into chunks (default: 0).
- **return_indices** (*bool, optional*) – If True, yield the array index that will return chunk rather than the chunk itself (default: False).

Yields *iterable* – Consecutive slices of *array* of size *n*.

nd.utils.**block_merge**(*array_list*, *blocks*)
Reassemble a list of arrays as generated by block_split.

Parameters

- **array_list** (*list of numpy.array*) – A list of numpy.array, e.g. as generated by block_split().
- **blocks** (*array_like*) – The number of blocks per axis to be merged.

Returns A numpy array with dimension len(blocks).

Return type numpy.array

nd.utils.**block_split**(*array*, *blocks*)
Split an ndarray into subarrays according to blocks.

Parameters

- **array** (*numpy.ndarray*) – The array to be split.
- **blocks** (*array_like*) – The desired number of blocks per axis.

Returns A list of blocks, in column-major order.

Return type list

Examples

```
>>> block_split(np.arange(16).reshape((4, 4)), (2, 2))
[array([[ 0,  1],
        [ 4,  5]]),
 array([[ 2,  3],
        [ 6,  7]]),
 array([[ 8,  9],
        [12, 13]]),
 array([[10, 11],
        [14, 15]])]
```

nd.utils.**chunks**(*l*, *n*)
Yield successive n-sized chunks from l.

https://stackoverflow.com/a/312464

Parameters

- **l** (*iterable*) – The list or list-like object to be split into chunks.
- **n** (*int*) – The size of the chunks to be generated.

Yields *iterable* – Consecutive slices of l of size n.

nd.utils.**dict_product**(*d*)
Like itertools.product, but works with dictionaries.

nd.utils.**expand_variables**(*da*, *dim='variable'*)
This is the inverse of xarray.Dataset.to_array().

Parameters

- **da** (*xarray.DataArray*) – A DataArray that contains the variable names as dimension.
- **dim** (*str*) – The dimension name (default: 'variable').

Returns A dataset with the variable dimension in *da* exploded to variables.

Return type xarray.Dataset

nd.utils.**get_dims**(*ds*)
Return the dimension of dataset *ds* in order.

nd.utils.**get_shape**(*ds*)

nd.utils.**get_vars_for_dims**(*ds*, *dims*, *invert=False*)
Return a list of all variables in *ds* which have dimensions *dims*.

Parameters

- **ds** (*xarray.Dataset*) –
- **dims** (*list of str*) – The dimensions that each variable must contain.
- **invert** (*bool, optional*) – Whether to return the variables that do *not* contain the given dimensions (default: False).

Returns A list of all variable names that have dimensions *dims*.

Return type list of str

nd.utils.**is_complex**(*ds*)
Check if a dataset contains any complex variables.

Parameters **ds** (*xarray.Dataset or xarray.DataArray*) –

Returns True if *ds* contains any complex variables, False otherwise.

Return type bool

nd.utils.**parallel**(*fn*, *dim=None*, *chunks=None*, *chunksize=None*, *merge=True*, *buffer=0*)
Parallelize a function that takes an xarray dataset as first argument.

TODO: make accept numpy arrays as well.

Parameters

- **fn** (*function*) – *Must* take an xarray.Dataset as first argument.
- **dim** (*str, optional*) – The dimension along which to split the dataset for parallel execution. If not passed, try 'y' as default dimension.
- **chunks** (*int, optional*) – The number of chunks to execute in parallel. If not passed, use the number of available CPUs.
- **chunksize** (*int, optional*) – ... to be implemented
- **buffer** (*int, optional*) – (default: 0)

Returns A parallelized function that may be called with exactly the same arguments as *fn*.

Return type function

nd.utils.**select**(*objects*, *fn*, *unlist=True*, *first=False*)
Returns a subset of *objects* that matches a range of criteria.

Parameters

- **objects** (*list of obj*) – The collection of objects to filter.
- **fn** (*lambda expression*) – Filter objects by whether fn(obj) returns True.
- **first** (*bool, optional*) – If True, return first entry only (default: False).
- **unlist** (*bool, optional*) – If True and the result has length 1 and objects is a list, return the object directly, rather than the list (default: True).

Returns A list of all items in *objects* that match the specified criteria.

Return type list

Examples

```
>>> select([{'a': 1, 'b': 2}, {'a': 2, 'b': 2}, {'a': 1, 'b': 1}],
            lambda o: o['a'] == 1)
[{'a': 1, 'b': 2}, {'a': 1, 'b': 1}]
```

nd.utils.**str2date**(*string*, *fmt=None*, *tz=False*)

nd.utils.**xr_merge**(*ds_list*, *dim*, *buffer=0*)
Reverse xr_split().

Parameters

- **ds_list** (*list of xarray.Dataset*) –
- **dim** (*str*) – The dimension along which to concatenate.

Returns

Return type xarray.Dataset

nd.utils.**xr_split**(*ds*, *dim*, *chunks*, *buffer=0*)
Split an xarray Dataset into chunks.

Parameters

- **ds** (*xarray.Dataset*) – The original dataset
- **dim** (*str*) – The dimension along which to split.
- **chunks** (*int*) – The number of chunks to generate.

Yields *xarray.Dataset* – An individual chunk.

3.8 nd.vector package

A submodule to deal with vector data.

nd.vector.**rasterize**(*shp*, *ds*, *columns=None*, *encode_labels=True*, *crs=None*, *date_field=None*, *date_fmt=None*)
Rasterize a vector dataset to match a reference raster.

Parameters

- **shp** (*str or geopandas.geodataframe.GeoDataFrame*) – Either the filename of a shapefile or an iterable
- **ds** (*xarray.Dataset*) – The reference dataset to match the raster shape.
- **columns** (*list of str, optional*) – List of column names to read.
- **encode_labels** (*bool, optional*) – If True, convert categorical data to integer values. The corresponding labels are accessible in the metadata. (default: True).
- **crs** (*str or dict or cartopy.crs.CRS, optional*) – The CRS of the vector data.
- **date_field** (*str, optional*) – The name of field containing the timestamp.
- **date_fmt** (*str, optional*) – The date format to parse date_field. Passed to *pd.to_datetime()*.

Returns The rasterized features.

Return type xarray.Dataset or xarray.DataArray

nd.vector.**read_file**(*path*, *clip=None*)
Read a geospatial vector file.

Parameters

- **path** (*str*) – The path of the file to read.
- **clip** (*shapely.geometry, optional*) – A geometry to intersect the vector data.

Returns

Return type geopandas.GeoDataFrame

3.9 nd.visualize package

Quickly visualize datasets.

nd.visualize.**colorize**(*labels*, *N=None*, *nan_vals=[]*, *cmap='jet'*)

Apply a color map to a map of integer labels.

Parameters

- **labels** (*np.array, shape (M,N)*) – The labeled image.
- **N** (*int, optional*) – The number of colors to use (default: 10)

Returns A colored image in BGR space, ready to be handled by OpenCV.

Return type np.array, shape (M,N,3)

nd.visualize.**plot_map**(*ds*, *buffer=None*, *background='_default'*, *imscale=6*, *gridlines=True*, *coastlines=True*, *scalebar=True*, *gridlines_kwargs={}*)

Show the boundary of the dataset on a visually appealing map.

Parameters

- **ds** (*xr.Dataset or xr.DataArray*) – The dataset whose bounds to plot on the map.
- **buffer** (*float, optional*) – Margin around the bounds polygon to plot, relative to the polygon dimension. By default, add around 20% on each side.
- **background** (`cartopy.io.img_tiles` image tiles, optional) – The basemap to plot in the background (default: Stamen terrain). If None, do not plot a background map.
- **imscale** (*int, optional*) – The zoom level of the background image (default: 6).
- **gridlines** (*bool, optional*) – Whether to plot gridlines (default: True).
- **coastlines** (*bool, optional*) – Whether to plot coastlines (default: True).
- **scalebar** (*bool, optional*) – Whether to add a scale bar (default: True).
- **gridlines_kwargs** (*dict, optional*) – Additional keyword arguments for gridlines_with_labels().

Returns The corresponding GeoAxes object.

Return type `cartopy.mpl.geoaxes.GeoAxes`

nd.visualize.**to_rgb**(*data*, *output=None*, *vmin=None*, *vmax=None*, *pmin=2*, *pmax=98*, *categorical=False*, *mask=None*, *shape=None*, *cmap=None*)

Turn some data into a numpy array representing an RGB image.

Parameters

- **data** (*list of DataArray*) –
- **output** (*str*) – file path
- **vmin** (*float or list of float*) – minimum value, or list of values per channel (default: None).
- **vmax** (*float or list of float*) – maximum value, or list of values per channel (default: None).
- **pmin** (*float*) – lowest percentile to plot (default: 2). Ignored if vmin is passed.
- **pmax** (*float*) – highest percentile to plot (default: 98). Ignored if vmax is passed.

- **categorical** (*bool, optional*) – Whether the data is categorical. If True, return a randomly colorized image according to the data value (default: False).
- **mask** (*np.ndarray, optional*) – If specified, parts of the image outside of the mask will be black.
- **shape** (*tuple, optional*) – The output height and width (either or both may be None)
- **cmap** (*opencv colormap, optional*) – The colormap used to colorize grayscale data

Returns Returns the generate RGB image if output is None, else returns None.

Return type np.ndarray or None

nd.visualize.**write_video**(*ds*, *path*, *timestamp='upper left'*, *fontcolor=(0, 0, 0)*, *width=None*, *height=None*, *fps=1*, *codec=None*, *rgb=None*, *cmap=None*, *mask=None*, *contours=None*, ***kwargs*)

Create a video from an xarray.Dataset.

Parameters

- **ds** (*xarray.Dataset or xarray.DataArray*) – The dataset must have dimensions 'y', 'x', and 'time'.
- **path** (*str*) – The output file path of the video.
- **timestamp** (*str, optional*) – Location to print the timestamp: ['upper left', 'lower left', 'upper right', 'lower right', 'ul', 'll', 'ur', 'lr'] Set to *None* to disable (default: 'upper left').
- **fontcolor** (*tuple, optional*) – RGB tuple for timestamp font color (default: (0, 0, 0), i.e., black).
- **width** (*int, optional*) – The width of the video (default: ds.dim['x'])
- **height** (*int, optional*) – The height of the video (default: ds.dim['y'])
- **fps** (*int, optional*) – Frames per second (default: 1).
- **codec** (*str, optional*) – fourcc codec (see http://www.fourcc.org/codecs.php)
- **rgb** (*callable, optional*) – A callable that takes a Dataset as input and returns a list of R, G, B channels. By default will compute the C11, C22, C11/C22 representation. For a DataArray, use `cmap`.
- **cmap** (*str, optional*) – For DataArrays only. Colormap used to colorize univariate data.
- **mask** (*np.ndarray, optional*) – If specified, parts of the image outside of the mask will be black.

3.10 nd.warp package

class nd.warp.**Alignment**(*target=None*, *crs=None*, *extent=None*)

Bases: `nd.algorithm.Algorithm`

Align a list of datasets to the same coordinate grid.

Parameters

- **target** (*xarray.Dataset, optional*) – Align the datasets with respect to the target dataset.
- **crs** (*str or dict, optional*) – The coordinate reference system as proj-string or dictionary. By default, use the CRS of the datasets.

- **extent** (*tuple, optional*) – The bounding box of the output dataset. By default, use the common extent of all datasets.

apply(*datasets*, *path*, ***, *njobs=1*)

Resample datasets to common extent and resolution.

Parameters

- **datasets** (*str, list of str, list of xarray.Dataset*) – The input datasets. Can be either a glob expression, a list of filenames, or a list of opened datasets.
- **path** (*str*) – The output path to store the aligned datasets.
- **njobs** (*int, optional*) – Number of jobs to run in parallel. Setting njobs to -1 uses the number of available cores. Disable parallelism by setting njobs to 1 (default).

class nd.warp.**Coregistration**(*reference=0*, *upsampling=10*)

Bases: nd.algorithm.Algorithm

Coregister a time series (stack) of images to a master image.

At the moment only supports coregistration by translation.

Parameters

- **reference** (*int, optional*) – The time index to use as reference for coregistration (default: 0).
- **upsampling** (*int, optional*) – The upsampling factor for shift estimation (default: 10).

apply(*ds*)

Apply the projection to a dataset.

Parameters **ds** (*xarray.Dataset*) – The input dataset.

Returns The coregistered dataset.

Return type xarray.Dataset

class nd.warp.**Reprojection**(*target=None*, *src_crs=None*, *dst_crs=None*, *crs=None*, *extent=None*, *res=None*, *width=None*, *height=None*, *transform=None*, ***kwargs*)

Bases: nd.algorithm.Algorithm

Reprojection of the dataset to the given coordinate reference system (CRS) and extent.

Parameters

- **target** (*xarray.Dataset or xarray.DataArray, optional*) – A reference to which a dataset will be aligned.
- **src_crs** (*dict or str, optional*) – The coordinate system of the input data (default: infer from data).
- **dst_crs** (*dict or str, optional*) – The output coordinate reference system as dictionary or proj-string
- **crs** (*dict or str, optional*) – Alias for dst_crs for backwards compatiblity.
- **extent** (*tuple, optional*) – The output extent. By default this is inferred from the input data.
- **res** (*tuple, optional*) – The output resolution. By default this is inferred from the input data.
- **width** (*tuple, optional*) – The output width. By default this is inferred from the input data.

- **height** (*tuple, optional*) – The output height. By default this is inferred from the input data.
- **transform** (*tuple, optional*) – The output coordinate transform. By default this is inferred from the input data.
- ****kwargs** (*dict, optional*) – Extra keyword arguments for `rasterio.warp.reproject`.

apply(*ds*, ***, *njobs=1*)

Apply the projection to a dataset.

Parameters

- **ds** (*xarray.Dataset*) – The input dataset.
- **njobs** (*int, optional*) – Number of jobs to run in parallel. Setting njobs to -1 uses the number of available cores. Disable parallelism by setting njobs to 1 (default).

Returns The reprojected dataset.

Return type xarray.Dataset

class nd.warp.**Resample**(*res=None*, *width=None*, *height=None*, ***kwargs*)

Bases: `nd.algorithm.Algorithm`

Resample a dataset to the specified resolution or width and height.

Parameters

- **res** (*float or tuple, optional*) – The desired resolution in the dataset coordinates.
- **width** (*int, optional*) – The desired output width. Ignored if the resolution is specified. If only the height is given, the width is calculated automatically.
- **height** (*int, optional*) – The desired output height. Ignored if the resolution is specified. If only the width is given, the height is calculated automatically.
- ****kwargs** (*dict, optional*) – Extra keyword arguments for `rasterio.warp.reproject`.

apply(*ds*, ***, *njobs=1*)

Resample the dataset.

Parameters

- **ds** (*xarray.Dataset or xarray.DataArray*) – The input dataset
- **njobs** (*int, optional*) – Number of jobs to run in parallel. Setting njobs to -1 uses the number of available cores. Disable parallelism by setting njobs to 1 (default).

Returns The resampled dataset.

Return type xarray.Dataset or xarray.DataArray

nd.warp.**align**(*datasets*, *path*, *target=None*, *crs=None*, *extent=None*, ***, *njobs=1*)

Wrapper for *nd.warp.Alignment*.

Align a list of datasets to the same coordinate grid.

Parameters

- **datasets** (*str, list of str, list of xarray.Dataset*) – The input datasets. Can be either a glob expression, a list of filenames, or a list of opened datasets.
- **path** (*str*) – The output path to store the aligned datasets.

- **target** (*xarray.Dataset, optional*) – Align the datasets with respect to the target dataset.
- **crs** (*str or dict, optional*) – The coordinate reference system as proj-string or dictionary. By default, use the CRS of the datasets.
- **extent** (*tuple, optional*) – The bounding box of the output dataset. By default, use the common extent of all datasets.
- **njobs** (*int, optional*) – Number of jobs to run in parallel. Setting njobs to -1 uses the number of available cores. Disable parallelism by setting njobs to 1 (default).

nd.warp.**coregister**(*ds*, *reference=0*, *upsampling=10*)

Wrapper for *nd.warp.Coregistration*.

Coregister a time series (stack) of images to a master image.

At the moment only supports coregistration by translation.

Parameters

- **ds** (*xarray.Dataset*) – The input dataset.
- **reference** (*int, optional*) – The time index to use as reference for coregistration (default: 0).
- **upsampling** (*int, optional*) – The upsampling factor for shift estimation (default: 10).

Returns The coregistered dataset.

Return type xarray.Dataset

nd.warp.**get_bounds**(*ds*)

Extract the bounding box in projection coordinates.

Parameters **ds** (*xarray.Dataset*) – The input dataset

Returns The bounding box in projection coordinates (left, bottom, right, top).

Return type tuple

nd.warp.**get_common_bounds**(*datasets*)

Calculate the common bounding box of the input datasets.

Parameters **datasets** (*list of xarray.Dataset*) – The input datasets.

Returns The common bounding box (left, bottom, right, top) in projected coordinates.

Return type tuple

nd.warp.**get_common_extent**(*datasets*)

Calculate the smallest extent that contains all of the input datasets.

Parameters **datasets** (*list of xarray.Dataset*) – The input datasets.

Returns The common extent (left, bottom, right, top) in latitude and longitude coordinates.

Return type tuple

nd.warp.**get_common_resolution**(*datasets*, *mode='min'*)

Determine the common resolution of a list of datasets.

Parameters

- **datasets** (*list of xarray.Dataset*) – The input datasets.
- **mode** (*str {'min', 'max', 'mean'}*) – How to determine the common resolution if the individual resolutions differ.

- `min`: Return the smallest (best) resolution.
- `max`: Return the largest (worst) resolution.
- `mean`: Return the average resolution.

Returns Returns the common resolution as (x, y).

Return type tuple

`nd.warp.get_crs`(*ds*, *format='crs'*)

Extract the Coordinate Reference System from a dataset.

Parameters

- **ds** (*xarray.Dataset*) – The input dataset
- **format** (*str {'crs', 'proj', 'dict', 'wkt'}*) – The format in which to return the CRS.
  - 'proj': A proj-string, e.g. `+init=epsg:4326`
  - 'dict': e.g. `{'init': 'EPSG:4326'}`
  - 'wkt': e.g. `GEOGCS["WGS 84", ...]`

Returns The CRS.

Return type CRS, str, or dict

`nd.warp.get_extent`(*ds*)

Extract the extent (bounding box) from the dataset.

Parameters **ds** (*xarray.Dataset*) – The input dataset

Returns The extent (left, bottom, right, top) in latitude and longitude coordinates.

Return type tuple

`nd.warp.get_geometry`(*ds*, *crs={'init': 'epsg:4326'}*)

Get the shapely geometry of the dataset bounding box in any coordinate system (EPSG:4326 by default).

Parameters

- **ds** (*xr.Dataset or xr.DataArray*) – The dataset whose geometry to return.
- **crs** (*dict, optional*) – The desired CRS as keywords arguments to be passed to `pyproj.Proj`.

Returns The bounds of the dataset in the desired coordinate system.

Return type shapely.geometry.Polygon

`nd.warp.get_resolution`(*ds*)

Extract the resolution of the dataset in projection coordinates.

Parameters **ds** (*xarray.Dataset*) – The input dataset

Returns The raster resolution as (x, y)

Return type tuple

`nd.warp.get_transform`(*ds*)

Extract the geographic transform from a dataset.

Parameters **ds** (*xarray.Dataset*) – The input dataset

Returns The affine transform

Return type affine.Affine

nd.warp.**ncols**(*ds*)

nd.warp.**nrows**(*ds*)

nd.warp.**reproject**(*ds*, *target=None*, *src_crs=None*, *dst_crs=None*, *crs=None*, *extent=None*, *res=None*, *width=None*, *height=None*, *transform=None*, ***, *njobs=1*, ***kwargs*)

Wrapper for *nd.warp.Reprojection*.

Reprojection of the dataset to the given coordinate reference system (CRS) and extent.

Parameters

- **ds** (*xarray.Dataset*) – The input dataset.
- **target** (*xarray.Dataset or xarray.DataArray, optional*) – A reference to which a dataset will be aligned.
- **src_crs** (*dict or str, optional*) – The coordinate system of the input data (default: infer from data).
- **dst_crs** (*dict or str, optional*) – The output coordinate reference system as dictionary or proj-string
- **crs** (*dict or str, optional*) – Alias for dst_crs for backwards compatiblity.
- **extent** (*tuple, optional*) – The output extent. By default this is inferred from the input data.
- **res** (*tuple, optional*) – The output resolution. By default this is inferred from the input data.
- **width** (*tuple, optional*) – The output width. By default this is inferred from the input data.
- **height** (*tuple, optional*) – The output height. By default this is inferred from the input data.
- **transform** (*tuple, optional*) – The output coordinate transform. By default this is inferred from the input data.
- **njobs** (*int, optional*) – Number of jobs to run in parallel. Setting njobs to -1 uses the number of available cores. Disable parallelism by setting njobs to 1 (default).
- ****kwargs** (*dict, optional*) – Extra keyword arguments for `rasterio.warp.reproject`.

Returns The reprojected dataset.

Return type xarray.Dataset

nd.warp.**resample**(*ds*, *res=None*, *width=None*, *height=None*, ***, *njobs=1*, ***kwargs*)

Wrapper for *nd.warp.Resample*.

Resample a dataset to the specified resolution or width and height.

Parameters

- **ds** (*xarray.Dataset or xarray.DataArray*) – The input dataset
- **res** (*float or tuple, optional*) – The desired resolution in the dataset coordinates.
- **width** (*int, optional*) – The desired output width. Ignored if the resolution is specified. If only the height is given, the width is calculated automatically.
- **height** (*int, optional*) – The desired output height. Ignored if the resolution is specified. If only the width is given, the height is calculated automatically.

- **njobs** (*int, optional*) – Number of jobs to run in parallel. Setting njobs to -1 uses the number of available cores. Disable parallelism by setting njobs to 1 (default).
- ****kwargs** (*dict, optional*) – Extra keyword arguments for `rasterio.warp.reproject`.

Returns The resampled dataset.

Return type xarray.Dataset or xarray.DataArray

3.11 `xarray` accessors

3.11.1 NDAccessor

This accessor provides direct access to much of the functionality of the whole `nd` package from `xarray` objects.

class nd._xarray.**NDAccessor**(*xarray_obj*)

Bases: `object`

apply(*fn*, *signature=None*, *njobs=1*)

Apply a function to a Dataset that operates on a defined subset of dimensions.

Parameters

- **fn** (*function*) – The function to apply to the Dataset.
- **signature** (*str, optional*) – The signature of the function in dimension names, e.g. '(time,var)->(time)'. If 'var' is included, the Dataset variables will be converted into a new dimension and the result will be a DataArray.
- **njobs** (*int, optional*) – The number of jobs to run in parallel.

Returns The output dataset with changed dimensions according to the function signature.

Return type xr.Dataset

as_complex(*inplace=False*)

Reassemble complex valued data.

NOTE: Changes the dataset (view) in place!

Parameters **inplace** (*bool, optional*) – Whether to modify the dataset inplace (default: False).

Returns If inplace, returns None. Otherwise, returns a dataset where the real and imaginary parts have been combined into the respective complex variables.

Return type xarray.Dataset or None

as_real(*inplace=False*)

Disassemble complex valued data into real and imag parts.

Parameters **inplace** (*bool, optional*) – Whether to modify the dataset inplace (default: False).

Returns If inplace, returns None. Otherwise, returns a dataset where all complex variables have been split into their real and imaginary parts.

Return type xarray.Dataset or None

change_omnibus(*ml=None*, *n=1*, *alpha=0.01*, **args*, ***kwargs*)
Wrapper for *nd.change.OmnibusTest*.

OmnibusTest

This class implements the change detection algorithm by Conradsen et al. (2015).

Parameters

- **ml** (*int, optional*) – Multilooking window size. By default, no multilooking is performed and the dataset is assumed to already be multilooked.
- **n** (*int, optional*) – The number of looks in *ds*. If *ml* is specified this parameter is ignored (default: 1).
- **alpha** (*float (0. ... 1.), optional*) – The significance level (default: 0.01).
- **kwargs** (*dict, optional*) – Extra keyword arguments to be applied to `ChangeDetection.__init__`.

plot_map(*buffer=None*, *background='_default'*, *imscale=6*, *gridlines=True*, *coastlines=True*, *scalebar=True*, *gridlines_kwargs={}*)
Show the boundary of the dataset on a visually appealing map.

Parameters

- **buffer** (*float, optional*) – Margin around the bounds polygon to plot, relative to the polygon dimension. By default, add around 20% on each side.
- **background** (`cartopy.io.img_tiles` image tiles, optional) – The basemap to plot in the background (default: Stamen terrain). If None, do not plot a background map.
- **imscale** (*int, optional*) – The zoom level of the background image (default: 6).
- **gridlines** (*bool, optional*) – Whether to plot gridlines (default: True).
- **coastlines** (*bool, optional*) – Whether to plot coastlines (default: True).
- **scalebar** (*bool, optional*) – Whether to add a scale bar (default: True).
- **gridlines_kwargs** (*dict, optional*) – Additional keyword arguments for gridlines_with_labels().

Returns The corresponding GeoAxes object.

Return type `cartopy.mpl.geoaxes.GeoAxes`

reproject(*target=None*, *src_crs=None*, *dst_crs=None*, *crs=None*, *extent=None*, *res=None*, *width=None*, *height=None*, *transform=None*, ***, *njobs=1*, ***kwargs*)
Wrapper for *nd.warp.Reprojection*.

Reprojection of the dataset to the given coordinate reference system (CRS) and extent.

Parameters

- **target** (*xarray.Dataset or xarray.DataArray, optional*) – A reference to which a dataset will be aligned.
- **src_crs** (*dict or str, optional*) – The coordinate system of the input data (default: infer from data).
- **dst_crs** (*dict or str, optional*) – The output coordinate reference system as dictionary or proj-string
- **crs** (*dict or str, optional*) – Alias for dst_crs for backwards compatiblity.

- **extent** (*tuple, optional*) – The output extent. By default this is inferred from the input data.
- **res** (*tuple, optional*) – The output resolution. By default this is inferred from the input data.
- **width** (*tuple, optional*) – The output width. By default this is inferred from the input data.
- **height** (*tuple, optional*) – The output height. By default this is inferred from the input data.
- **transform** (*tuple, optional*) – The output coordinate transform. By default this is inferred from the input data.
- **njobs** (*int, optional*) – Number of jobs to run in parallel. Setting njobs to -1 uses the number of available cores. Disable parallelism by setting njobs to 1 (default).
- ****kwargs** (*dict, optional*) – Extra keyword arguments for `rasterio.warp.reproject`.

Returns The reprojected dataset.

Return type xarray.Dataset

resample(*res=None*, *width=None*, *height=None*, ***, *njobs=1*, ***kwargs*)
Wrapper for *nd.warp.Resample*.

Resample a dataset to the specified resolution or width and height.

Parameters

- **res** (*float or tuple, optional*) – The desired resolution in the dataset coordinates.
- **width** (*int, optional*) – The desired output width. Ignored if the resolution is specified. If only the height is given, the width is calculated automatically.
- **height** (*int, optional*) – The desired output height. Ignored if the resolution is specified. If only the width is given, the height is calculated automatically.
- **njobs** (*int, optional*) – Number of jobs to run in parallel. Setting njobs to -1 uses the number of available cores. Disable parallelism by setting njobs to 1 (default).
- ****kwargs** (*dict, optional*) – Extra keyword arguments for `rasterio.warp.reproject`.

Returns The resampled dataset.

Return type xarray.Dataset or xarray.DataArray

to_netcdf(*path*, **args*, ***kwargs*)
Write an xarray Dataset to disk.

In addition to `xarray.to_netcdf`, this function allows to store complex valued data by converting it to a a pair of reals. This process is reverted when reading the file via `from_netcdf`.

Parameters

- **path** (*str*) – The path of the target NetCDF file.
- ***args** (*list*) – Extra positional arguments for `xr.Dataset.to_netcdf`.
- ****kwargs** (*dict*) – Extra keyword arguments for `xr.Dataset.to_netcdf`.

to_rgb(*output=None*, *vmin=None*, *vmax=None*, *pmin=2*, *pmax=98*, *categorical=False*, *mask=None*, *shape=None*, *cmap=None*, *rgb=None*)

Turn some data into a numpy array representing an RGB image.

Parameters

- **output** (*str*) – file path
- **vmin** (*float or list of float*) – minimum value, or list of values per channel (default: None).
- **vmax** (*float or list of float*) – maximum value, or list of values per channel (default: None).
- **pmin** (*float*) – lowest percentile to plot (default: 2). Ignored if vmin is passed.
- **pmax** (*float*) – highest percentile to plot (default: 98). Ignored if vmax is passed.
- **categorical** (*bool, optional*) – Whether the data is categorical. If True, return a randomly colorized image according to the data value (default: False).
- **mask** (*np.ndarray, optional*) – If specified, parts of the image outside of the mask will be black.
- **shape** (*tuple, optional*) – The output height and width (either or both may be None)
- **cmap** (*opencv colormap, optional*) – The colormap used to colorize grayscale data
- **rgb** (*callable*) – A function returning an RGB tuple from the dataset, e.g., *lambda d: [d.B4, d.B3, d.B2]* or *lambda d: [d.isel(band=0), d.isel(band=1), d.isel(band=2)]*

Returns Returns the generate RGB image if output is None, else returns None.

Return type np.ndarray or None

to_video(*path*, *timestamp='upper left'*, *fontcolor=(0, 0, 0)*, *width=None*, *height=None*, *fps=1*, *codec=None*, *rgb=None*, *cmap=None*, *mask=None*, *contours=None*, ***kwargs*)

Create a video from an xarray.Dataset.

Parameters

- **path** (*str*) – The output file path of the video.
- **timestamp** (*str, optional*) – Location to print the timestamp: [‘upper left’, ‘lower left’, ‘upper right’, ‘lower right’, ‘ul’, ‘ll’, ‘ur’, ‘lr’] Set to *None* to disable (default: ‘upper left’).
- **fontcolor** (*tuple, optional*) – RGB tuple for timestamp font color (default: (0, 0, 0), i.e., black).
- **width** (*int, optional*) – The width of the video (default: ds.dim[‘x’])
- **height** (*int, optional*) – The height of the video (default: ds.dim[‘y’])
- **fps** (*int, optional*) – Frames per second (default: 1).
- **codec** (*str, optional*) – fourcc codec (see http://www.fourcc.org/codecs.php)
- **rgb** (*callable, optional*) – A callable that takes a Dataset as input and returns a list of R, G, B channels. By default will compute the C11, C22, C11/C22 representation. For a DataArray, use `cmap`.
- **cmap** (*str, optional*) – For DataArrays only. Colormap used to colorize univariate data.

- **mask** (*np.ndarray, optional*) – If specified, parts of the image outside of the mask will be black.

3.11.2 FilterAccessor

This separate accessor provides direct access to the filters implemented in `nd.filters`.

class nd._xarray.**FilterAccessor**(*xarray_obj*)
Bases: object

boxcar(*inplace=False*, *dims=('y', 'x')*, *w=3*, ***, *njobs=1*, ***kwargs*)
Wrapper for *nd.filters.BoxcarFilter*.

A boxcar filter.

Parameters

- **inplace** (*bool, optional*) – If True, overwrite the input data inplace (default: False).
- **dims** (*tuple of str, optional*) – The dimensions along which to apply the filter (default: ('y', 'x')).
- **w** (*int*) – The width of the boxcar window. Should be an odd integer in order to ensure symmetry.
- **njobs** (*int, optional*) – Number of jobs to run in parallel. Setting njobs to -1 uses the number of available cores. Disable parallelism by setting njobs to 1 (default).
- **kwargs** (*dict, optional*) – Extra keyword arguments passed on to `scipy.ndimage.filters.convolve`.

Returns The filtered dataset

Return type xarray.Dataset

convolve(*inplace=False*, *dims=('y', 'x')*, *kernel=None*, ***, *njobs=1*, ***kwargs*)
Wrapper for *nd.filters.ConvolutionFilter*.

Kernel-convolution of an xarray.Dataset.

Parameters

- **inplace** (*bool, optional*) – If True, overwrite the input data inplace (default: False).
- **dims** (*tuple, optional*) – The dataset dimensions corresponding to the kernel axes (default: ('y', 'x')). The length of the tuple must match the number of dimensions of the kernel.
- **kernel** (*ndarray*) – The convolution kernel.
- **njobs** (*int, optional*) – Number of jobs to run in parallel. Setting njobs to -1 uses the number of available cores. Disable parallelism by setting njobs to 1 (default).
- **kwargs** (*dict, optional*) – Extra keyword arguments passed on to `scipy.ndimage.filters.convolve`.

Returns The filtered dataset

Return type xarray.Dataset

gaussian(*inplace=False*, *dims=('y', 'x')*, *sigma=1*, ***, *njobs=1*, ***kwargs*)
Wrapper for *nd.filters.GaussianFilter*.

A Gaussian filter.

Parameters

- **inplace** (*bool, optional*) – If True, overwrite the input data inplace (default: False).
- **dims** (*tuple of str, optional*) – The dimensions along which to apply the Gaussian filtering (default: ('y', 'x')).
- **sigma** (*float or sequence of float*) – The standard deviation for the Gaussian kernel. If sequence, this is set individually for each dimension.
- **njobs** (*int, optional*) – Number of jobs to run in parallel. Setting njobs to -1 uses the number of available cores. Disable parallelism by setting njobs to 1 (default).
- **kwargs** (*dict, optional*) – Extra keyword arguments passed on to `scipy.ndimage.filters.gaussian_filter`.

Returns The filtered dataset

Return type xarray.Dataset

nlmeans(*inplace=False*, *dims=('y', 'x')*, *r=1*, *sigma=1*, *h=1*, *f=1*, *n_eff=- 1*, ***, *njobs=1*)

Wrapper for *nd.filters.NLMeansFilter*.

Non-Local Means (Buades2011).

Buades, A., Coll, B., & Morel, J.-M. (2011). Non-Local Means Denoising. Image Processing On Line, 1, 208–212. https://doi.org/10.5201/ipol.2011.bcm_nlm

Parameters

- **inplace** (*bool, optional*) – If True, overwrite the input data inplace (default: False).
- **dims** (*tuple of str*) – The dataset dimensions along which to filter.
- **r** (*{int, sequence}*) – The radius
- **sigma** (*float*) – The standard deviation of the noise present in the data.
- **h** (*float*) –
- **f** (*int*) –
- **n_eff** (*float, optional*) – The desired effective sample size. If given, must be greater than 1 and should be no larger than about half the pixels in the window. -1 means no fixed effective sample size (default: -1).
- **njobs** (*int, optional*) – Number of jobs to run in parallel. Setting njobs to -1 uses the number of available cores. Disable parallelism by setting njobs to 1 (default).

Returns The filtered dataset

Return type xarray.Dataset

PYTHON MODULE INDEX

n

nd.change, 21
nd.classify, 22
nd.filters, 23
nd.io, 28
nd.tiling, 30
nd.utils, 32
nd.vector, 35
nd.visualize, 36
nd.warp, 37

INDEX

Symbols

_expand_kernel() (*in module nd.filters*), 23

A

add_time() (*in module nd.io*), 28
align() (*in module nd.warp*), 39
Alignment (*class in nd.warp*), 37
apply() (*in module nd.utils*), 32
apply() (*nd._xarray.NDAccessor method*), 43
apply() (*nd.change.OmnibusTest method*), 21
apply() (*nd.filters.BoxcarFilter method*), 24
apply() (*nd.filters.ConvolutionFilter method*), 24
apply() (*nd.filters.Filter method*), 25
apply() (*nd.filters.GaussianFilter method*), 25
apply() (*nd.filters.NLMeansFilter method*), 26
apply() (*nd.warp.Alignment method*), 38
apply() (*nd.warp.Coregistration method*), 38
apply() (*nd.warp.Reprojection method*), 39
apply() (*nd.warp.Resample method*), 39
array_chunks() (*in module nd.utils*), 32
as_complex() (*nd._xarray.NDAccessor method*), 43
as_real() (*nd._xarray.NDAccessor method*), 43
assemble_complex() (*in module nd.io*), 28
auto_merge() (*in module nd.tiling*), 30

B

block_merge() (*in module nd.utils*), 32
block_split() (*in module nd.utils*), 32
boxcar() (*in module nd.filters*), 26
boxcar() (*nd._xarray.FilterAccessor method*), 47
BoxcarFilter (*class in nd.filters*), 23

C

change_omnibus() (*nd._xarray.NDAccessor method*), 43
ChangeDetection (*class in nd.change*), 21
chunks() (*in module nd.utils*), 33
class_mean() (*in module nd.classify*), 23
Classifier (*class in nd.classify*), 22
colorize() (*in module nd.visualize*), 36
convolution() (*in module nd.filters*), 27
ConvolutionFilter (*class in nd.filters*), 24
convolve() (*nd._xarray.FilterAccessor method*), 47
coregister() (*in module nd.warp*), 40
Coregistration (*class in nd.warp*), 38

D

debuffer() (*in module nd.tiling*), 30
dict_product() (*in module nd.utils*), 33
dims (*nd.filters.BoxcarFilter attribute*), 24
dims (*nd.filters.ConvolutionFilter attribute*), 24
dims (*nd.filters.Filter attribute*), 25
dims (*nd.filters.GaussianFilter attribute*), 25
dims (*nd.filters.NLMeansFilter attribute*), 26
disassemble_complex() (*in module nd.io*), 28

E

expand_variables() (*in module nd.utils*), 33

F

Filter (*class in nd.filters*), 24
FilterAccessor (*class in nd._xarray*), 47
fit() (*nd.classify.Classifier method*), 22
fit_predict() (*nd.classify.Classifier method*), 22

G

gaussian() (*in module nd.filters*), 27
gaussian() (*nd._xarray.FilterAccessor method*), 47
GaussianFilter (*class in nd.filters*), 25
get_bounds() (*in module nd.warp*), 40
get_common_bounds() (*in module nd.warp*), 40
get_common_extent() (*in module nd.warp*), 40
get_common_resolution() (*in module nd.warp*), 40
get_crs() (*in module nd.warp*), 41
get_dims() (*in module nd.utils*), 33
get_extent() (*in module nd.warp*), 41
get_geometry() (*in module nd.warp*), 41
get_resolution() (*in module nd.warp*), 41
get_shape() (*in module nd.utils*), 33
get_transform() (*in module nd.warp*), 41
get_vars_for_dims() (*in module nd.utils*), 33

I

is_complex() (*in module nd.utils*), 33

K

kwargs (*nd.filters.BoxcarFilter attribute*), 24
kwargs (*nd.filters.ConvolutionFilter attribute*), 24

M

make_Xy() (*nd.classify.Classifier method*), 22
map_over_tiles() (*in module nd.tiling*), 30
module
    nd.change, 21
    nd.classify, 22
    nd.filters, 23
    nd.io, 28
    nd.tiling, 30
    nd.utils, 32
    nd.vector, 35
    nd.visualize, 36
    nd.warp, 37

N

ncols() (*in module nd.warp*), 41
nd.change
    module, 21
nd.classify
    module, 22
nd.filters
    module, 23
nd.io
    module, 28
nd.tiling
    module, 30
nd.utils
    module, 32
nd.vector
    module, 35
nd.visualize
    module, 36
nd.warp
    module, 37
NDAccessor (*class in nd._xarray*), 43
njobs (*nd.change.ChangeDetection attribute*), 21
nlmeans() (*in module nd.filters*), 27
nlmeans() (*nd._xarray.FilterAccessor method*), 48
NLMeansFilter (*class in nd.filters*), 26
nrows() (*in module nd.warp*), 42

O

omnibus() (*in module nd.change*), 21
OmnibusTest (*class in nd.change*), 21
open_beam_dimap() (*in module nd.io*), 29
open_dataset() (*in module nd.io*), 29
open_netcdf() (*in module nd.io*), 29
open_rasterio() (*in module nd.io*), 29

P

parallel() (*in module nd.utils*), 34
per_variable (*nd.filters.BoxcarFilter attribute*), 24
per_variable (*nd.filters.ConvolutionFilter attribute*), 24
per_variable (*nd.filters.Filter attribute*), 25
per_variable (*nd.filters.GaussianFilter attribute*), 25
per_variable (*nd.filters.NLMeansFilter attribute*), 26
plot_map() (*in module nd.visualize*), 36
plot_map() (*nd._xarray.NDAccessor method*), 44
predict() (*nd.classify.Classifier method*), 22

R

rasterize() (*in module nd.vector*), 35
read_file() (*in module nd.vector*), 35
reproject() (*in module nd.warp*), 42
reproject() (*nd._xarray.NDAccessor method*), 44
Reprojection (*class in nd.warp*), 38
Resample (*class in nd.warp*), 39
resample() (*in module nd.warp*), 42
resample() (*nd._xarray.NDAccessor method*), 45

S

score() (*nd.classify.Classifier method*), 22
select() (*in module nd.utils*), 34
sort_into_array() (*in module nd.tiling*), 31
sort_key() (*in module nd.tiling*), 31
str2date() (*in module nd.utils*), 34
supports_complex (*nd.filters.BoxcarFilter attribute*), 24
supports_complex (*nd.filters.ConvolutionFilter attribute*), 24
supports_complex (*nd.filters.Filter attribute*), 25
supports_complex (*nd.filters.GaussianFilter attribute*), 25
supports_complex (*nd.filters.NLMeansFilter attribute*), 26

T

tile() (*in module nd.tiling*), 31
to_netcdf() (*in module nd.io*), 30
to_netcdf() (*nd._xarray.NDAccessor method*), 45
to_rgb() (*in module nd.visualize*), 36
to_rgb() (*nd._xarray.NDAccessor method*), 45
to_video() (*nd._xarray.NDAccessor method*), 46

W

write_video() (*in module nd.visualize*), 37

X

xr_merge() (*in module nd.utils*), 34

xr_split() (*in module nd.utils*), 35