Author(s): Annala, Leevi; Eskelinen, Matti; Hämäläinen, Jyri; Riihinen, Aamos; Pölönen, Ilkka

Title: Practical Approach for Hyperspectral Image Processing in Python

Year: 2018

Version: Published version

Please cite the original version:

Annala, L., Eskelinen, M., Hämäläinen, J., Riihinen, A., & Pölönen, I. (2018). Practical Approach for Hyperspectral Image Processing in Python. In J. Jiang, A. Shaker, H. Zhang, X. Liang, B. Osmanoglu, U. Soergel, E. Honkavaara, M. Scaioni, J. Zhang, A. Peled, L. Wu, R. Li, M. Yoshimura, K. Di, T. J. Tanzi, H. M. Abdulmuttalib, F. S. Faruque, U. Stilla, & K. Komp (Eds.), ISPRS TC III Mid-term Symposium "Developments, Technologies and Applications in Remote Sensing" (pp. 45-52). International Society for Photogrammetry and Remote Sensing. International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XLII-3. https://doi.org/10.5194/isprs-archives-XLII-3-45-2018

PRACTICAL APPROACH FOR HYPERSPECTRAL IMAGE PROCESSING IN PYTHON

Leevi Annala*, Matti A. Eskelinen, Jyri Hämäläinen, Aamos Riihinen, Ilkka Pölönen

Faculty of Information Technology, University of Jyvaskyla - leevi.a.annala@student.jyu.fi

Commission III, WG III/4

KEY WORDS: Python, Data analysis, Hyperspectral imaging, Image processing, Machine learning, Open source

ABSTRACT:

Python is a very popular programming language among data scientists around the world. Python can also be used in hyperspectral data analysis. There are some toolboxes designed for spectral imaging, such as Spectral Python and HyperSpy, but there is a need for analysis pipeline, which is easy to use and agile for different solutions. We propose a Python pipeline which is built on packages xarray, Holoviews and scikit-learn. We have developed some of own tools, MaskAccessor, VisualisorAccessor and a spectral index library. They also fulfill our goal of easy and agile data processing. In this paper we will present our processing pipeline and demonstrate it in practice.

1. INTRODUCTION AND MOTIVATION

Python is a go-to programming language of many scientists and it could also be good programming language for hyperspectral data analysis. It has advantage of being actively developed, free, open source programming language. In addition, since it looks like pseudocode, it is easy to learn and write. There are Python tools and packages for all kinds of users, and especially for scientists. There are specialized open source tools for hyperspectral data analysis like Spectral Python (Boggs, n.d.) and HyperSpy (de la Peña et al., 2017), but the scope of potential usage may be too narrow and the structure of such an specialized tool can be too strict for some purposes, for example for transferring data to machine learning algorithm and developing tools that work together with them.

In this paper, we utilize some general open source tools for different aspects of hyperspectral data analysis and determine if they are useful for analysing and visualising hyperspectral images. We also introduce some new tools and packages, which are our own work. We aim at providing the reader with a modular set of tools that can be used in many contexes. These tools are reusable elements, which work fine on their own and can be used for building more complex tools. The packages and tools will be evaluated using following questions: How easy is it to use? How agile is it? What can we do with it?

2. DIFFERENT ASPECTS OF HYPERSPECTRAL DATA ANALYSIS

In this section we will go through different aspects of hyperspectral data analysis and an example of how the selected tools can be used in these subjects. The example is divided into smaller examples and what has been done on previously is assumed to hold on to the new example. We go through the example in figures and in text, and the source code is included in the figures. The example problem is that we have a hyperspectral image of a forest and a dataset of two tree species, birch and pine, in that forest, and we want to use machine learning to differentiate one from the other. First we of course need to import all of the packages, like in figure 1.

```
import xarray as xr
import numpy as np
import pandas as pd
import holoviews as hv
from sklearn import svm
import sklearn
from sklearn.model_selection import GridSearchCV
import visacc
import maskacc

hv.notebook_extension('matplotlib')
```

Figure 1. Importing all necessary packages and declaring that Holoviews should use Matplotlib backend.

2.1 Handling hyperspectral data

For handling hyperspectral data, we recommend the xarray[1] package (Hoyer and Hamman, 2017). It provides multidimensional arrays and datasets with metadata. It is an actively developed open source project by the pydata team. The basic usage of xarray is relatively easy and for more advanced users it offers plenty of options for handling the data. Xarray’s basic idea is to have netCDF (Rew et al., 1997) compatible multidimensional array object in Python. NetCDF stands for network Common Data Form and the basic idea is that the netCDF file describes itself to the reader. Xarray is also easily extendable, which means that one can add new properties as they are needed.

Xarray supports reading spectral image formats like ENVI or TIFF, and other formats. For reading it uses Rasterio (Gillies et al., 2013–), which in turn uses GDAL (GDAL Development Team, 2018). Rasterio is a python toolbox developed solely to read and write geospatial data, and it does it well. GDAL (Geospatial Data Abstraction Library) is a lower level C++ library that translates geospatial raster and vector data.

When xarray has read dataset from file (see figure 2), it is either DataArray or Dataset. There are differences between the two, but

*Corresponding author

[1] Xarray can be installed with pip (`pip install xarray`) or conda (`conda install xarray`) Python package managers.

from now on we will assume that the data is in DaraArray format. DataArray has following properties (see figure 3):

- data, N-dimensional NumPy (Oliphant, 2006) or Dask (Dask Development Team, 2016) array,
- coords, dictionary of coordinate arrays, one array for each dimension of the data,
- dims, names of the dimensions,
- attrs, dictionary keeping track of other metadata,
- name, the name of the DataArray,

which follow the netCDF specification. These properties help in

```
cube = xr.open_dataarray(
        'C:/Users/lealanna/DATAA/vvkk2.nc'
        )
wavelength = [507.60, 509.50, 514.50, 520.80,
              529.00, 537.40, 545.80, 554.40,
              562.70, 574.20, 583.60, 590.40,
              598.80, 605.70, 617.50, 630.70,
              644.20, 657.20, 670.10, 677.80,
              691.10, 698.40, 705.30, 711.10,
              717.90, 731.30, 738.50, 751.50,
              763.70, 778.50, 794.00, 806.30,
              819.70, 833.70, 845.80, 859.10,
              872.80, 885.60]
cube.coords['wavelength'] = ('band', wavelength)
cube = cube.swap_dims({'band': 'wavelength'})
cube.values[cube.values<0]=np.nan
```

Figure 2. Here we read the cube, attach wavelength data to it and remove non-physical negative values.

```
print(cube)

<xarray.DataArray (wavelength: 38, y: 4120, x: 3930)>
array([[[ nan,  nan, ...,  nan,  nan],
        [ nan,  nan, ...,  nan,  nan],
        ...,
        [ nan,  nan, ...,  nan,  nan],
        [ nan,  nan, ...,  nan,  nan]],

        ...,
        [ nan,  nan, ...,  nan,  nan],
        [ nan,  nan, ...,  nan,  nan]],
        ...,
        [ nan,  nan, ...,  nan,  nan],
        [ nan,  nan, ...,  nan,  nan]]],
        dtype=float32)
Coordinates:
    * longitude    (longitude) float64 6.804e+06 ...
    * latitude     (latitude) float64 3.983e+05  ...
    band          (wavelength) int32 1 2 3 4 5 6 7 ...
    * wavelength   (wavelength) float64 507.6 509.5...
Attributes:
    res:          [ 1. -1.]
    is_tiled:     1
    transform:    [1.00000000e-01  0.00000000e+00 ...
    ncols:        3930
    rows:         4120
    xllcorner:    398296
    yllcorner:    6804299
    cellsize:     0.1
```

Figure 3. Simple print-command to see what the cube holds inside.

extracting data from the DataArray, since the user can use either index based lookups or label based lookups. For example, if we only had NumPy[2] array, we would only know the dimensions by index, but with DataArray we have names like latitude, longitude and wavelength[3]. Then we can extract data from DataArray like in figure 4 by telling it that we want to see data where latitude is between 39° N and 40° N, longitude is between 116° E and 117° E, and wavelength is between 400 nm and 700 nm.

```
cube.sel(latitude=slice(39,40),
         longitude=slice(116,117),
         wavelength=slice(400,700))
```

Figure 4. Here we use xarray's sel-method to extract the data we want.

There are also other useful functionalities of xarray DataArray. For example two or more arrays can be attached to each other with easy one line command, where the user only has to align the arrays by common dimension. Generally speaking, xarray handles dimensions well and altering and extracting data using them is generally quite easy. Xarray also handles missing data well and there is possibility to use Dask arrays to parallel compute.

Xarray fullfills our criteria of being easy to use and agile. It has a lot of functionality, enough to keep basic and advanced users satisfied most of the time.

2.2 Visualisation

For visualizing the xarray data, one excellent solution is Holoviews[4] (Stevens et al., n.d.). Holoviews is a visualization library that uses Bokeh (Bokeh Development Team, 2014), Matplotlib (Hunter, 2007) or Plotly (Plotly Technologies Inc., 2015) for showing images. All figures in this paper are produced with Holoviews using Bokeh or Matplotlib visualisation backends.

Basic idea of Holoviews is that visualizing of data should be easy and simple. If user wants to see anything, it should not take many lines of code. In our opinion, Holoviews succeeds in that goal. As we move on, one will see that all images in this paper are produced with less than four lines of code. One basic example of producing Holoviews image is to look at one band of a hyperspectral image like in figure 5.

Now that we have figured out how to visualise a single channel of an image, the next logical step is to want to visualise the entire multidimensional dataset. This is also easy. Holoviews supports multidimensional datasets very well and there are data backends that support multiple different data formats including xarray. As we can see in figure 6, more complex visualisation is easy to make. In the example we make a Holoviews dataset out of xarray DataArray, and tell Holoviews to make a series of images out of the dataset.

One of the properties of Holoviews is that one can make interactive figures using the Bokeh backend with no extra effort. By having Bokeh backend selected user can right away use interactive tools like zooming the image either by scrolling or drawing boxes on the image. A little more work is required for using hover, tapping or selection tools, which all can be programmed to do what the user wants them to do. An example of usage of tapping and selection tools are using them to select data for further analysis or activating other visualisation with them.

[2]NumPy is in practice the Python standard array library.

[3]Note, that the user can freely name the dimensions. The user is not stuck with these names.

[4]Holoviews can be installed with pip (`pip install holoviews`) or conda (`conda install holoviews`) Python package managers.

```
cube = xarray.DataArray(...)
hv.Image(cube.sel(wavelength=800,
                  method='nearest'))
```

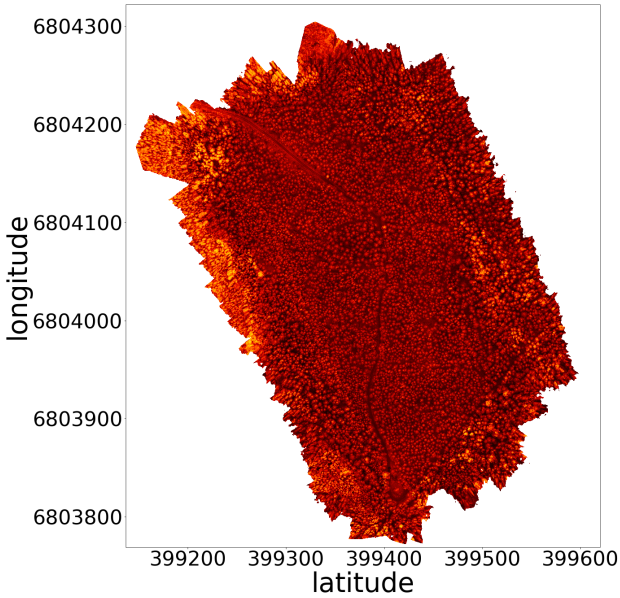

Figure 5. Here we produce a very simple Holoviews visualisation by telling Holoviews Image to use the xarray data we provide it. This is an image of a Finnish forest.

A good platform for using Holoviews is Jupyter Notebook (Kluyver et al., 2016). Jupyter Notebook is a web application where user can code in Python and output images and write narratives between code blocks. The user has to activate Holoviews by importing it and declaring the visualisation backend like in figure 1. When one is visualising figures in Jupyter Notebook, it is possible to fine tune figures, by using *output cell magic* and Holoviews *opts*. We use output cell magic and opts in figure 6, where lines starting with % or %% are the cell magic lines. In the example of the figure we tune the size of the font and the size of the image. This fine tuning is absolutely necessary if one needs to produce figures for a publication or needs good looking images for any reason. Matplotlib backend is better suited for publication quality figures.

Holoviews is purely a visualisation library. The user can make data move between two images in the same visualisation, but the developers have not build a way to get this data for further use and the only way of getting a data output is by coding it. However, once coded, these background processes are relatively easy to attach to an image. Holoviews is an open source project and it is developed by the ioam team. Holoviews is easy to use and it can be bended to do many things. It makes beautiful images, and in all is an excellent choise for visualisation.

2.3 Masking and visualizing xarray

Using xarray and Holoviews together is made easy by Holoviews developers. Xarray is one of the available backends for Holoviews. That means, one can easily produce an image from xarray using Holoviews. There is still some difficulties involved, and to address those difficulties, we use xarrays extendability. Making an extension to xarray in figure 7 is done by making a Python class and declaring it as dataset or DataArray accessor.

```
%%opts Image [fontsize={{'title':15, \
                         'xlabel':15, \
                         'ylabel':15, \
                         'ticks':15}},\
              fig_size=350]

ds = hv.Dataset(cube,
                vdims=['Value'])
ds.to(hv.Image,
      kdims=['x', 'y'],
      dynamic=True)
```

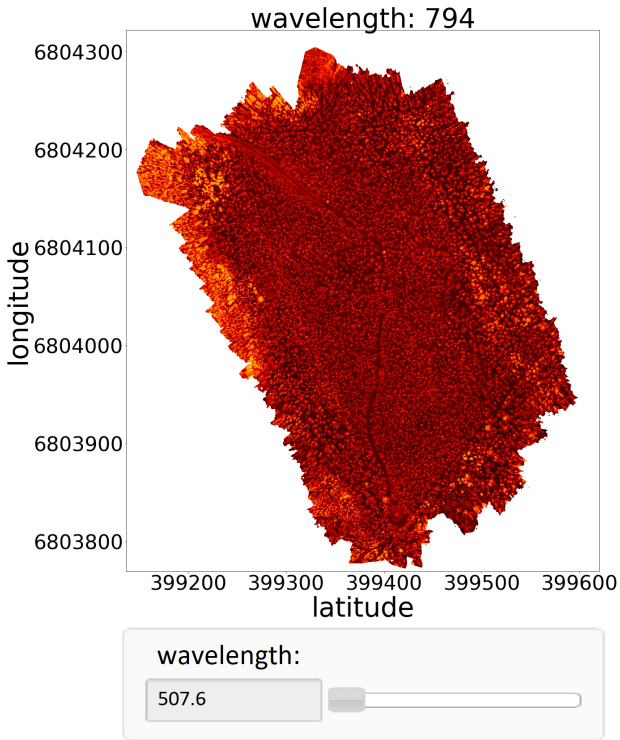

Figure 6. Here we make a more complicated Holoviews visualisation by using Holoviews dataset. From using Dataset, we get a slider that goes through the wavelength bands.

These extensions are relatively easy to make and can extend xarray's functionalities to anything one might want it to do, within reasonable limits.

```
@xr.register_DataArray_accessor('cat')
class CatAccessor(object):
    def __init__(self, xarray_obj):
        self._obj = xarray_obj
        self.cat = 'a_cat'
```

Figure 7. Extending xarray with a simple Accessor. Here we declare that CatAccessor is a DataArray accessor and define it.

We have developed two DataArray accessors, MaskAccessor and VisualisorAccessor[5]. The reason for developing both of these tools is that we want to use more complicted background interactivity tracking with Holoviews and get the data out of the visualisation.

MaskAccessor is a general masking tool for xarray, and the main function of it is to help collect data points to further analysis, such as machine learning or modelling. It provides an interface

[5]These tools are available at our groups github page `http://github.com/silmae`

for selecting pixels in n-dimensional datasets. In figure 8 we see that the accessor is initiated when one imports the xarray and the accessor. After that every DataArray has the property, and the user can use the accessor by calling it by name.

```
import xarray as xr
import maskacc

cube = xr.DataArray(...)
cube.M.dims
```

Figure 8. When the accessor is imported, every xarray DataArray created after that has the accessor attribute.

The mask dimensions are set at the initialisation to be the first two dimensions of the DataArray, but there is the reset method that is used to change the dimensions, as we see on figure 9. One can also initialise the mask here or just assign a new mask afterwards. The MaskAccessor class checks that the shape of the mask is correct.

```
import numpy as np
cube.M.reset(dims=['a', 'b'],
             matrix=[[0,1,0,1],
                     [1,0,1,0],
                     [0,1,0,1]])

# OR
cube.M.reset(dims=['a', 'b'])
cube.M.mask = np.array([[0,1,0,1],
                        [1,0,1,0],
                        [0,1,0,1]])
```

Figure 9. Different ways of assigning a specific matrix as the mask.

On figure 10 one can see four different selection methods to set mask on individual points.

```
# Select
cube.M.select([0,0])
cube.M.select([(0,2),(1,1)])

# Unselect
cube.M.unselect([(0,2),(1,1)])

# All to ones
cube.M.selected_ones()

# All to zeros
cube.M.selected_zeros()
```

Figure 10. Different selection methods for MaskAccessor.

Finally, on figure 11 there is three different methods to get the mask or masked data.

```
# Get the mask as xarray.DataArray
cube.M.mask_as_xarray()

# Get the masked points as xarray.DataArray
cube.M.where_masked()

# Get the masked points as a list
cube.M.to_list()
```

Figure 11. Methods for getting data out of MaskAccessor and underlyind DataArray.

VisualisorAccessor is a hyperspectral imaging specific visualising tool for xarray and MaskAccessor. It is designed to make basic visualizations of xarray DataArray and MaskAccessor mask with easy one-line commands. For example the image in figure 6 can now be produced with the one line code of figure 12. It is also easy to add visualisations like this to the VisualisorAccessor.

```
cube.visualize.basic(sliders = ['wavelength'])
```

Figure 12. The visualisation on figure 6 can be done with one line code with VisualisorAccessor.

We have implemented three chooser functions, which access the mask and select or unselect pixels. They are called Point Chooser, Box Chooser and Spectre Chooser. Spectre Chooser and Box Chooser use Bokeh’s box drawing tools for selecting which pixels are chosen and Point Chooser uses tap tool. Example uses of the Choosers is on figure 13, and screenshots of the Choosers are on figures 14 (Point Chooser), 15 (Box Chooser) and 16 (Spectre Chooser).

```
layout_box = cube.visualize.box_chooser()
layout_point = cube.visualize.point_chooser()
layout_spectre = cube.visualize.spectre_chooser()
```

Figure 13. VisualisorAccessor has three different chooser tools.

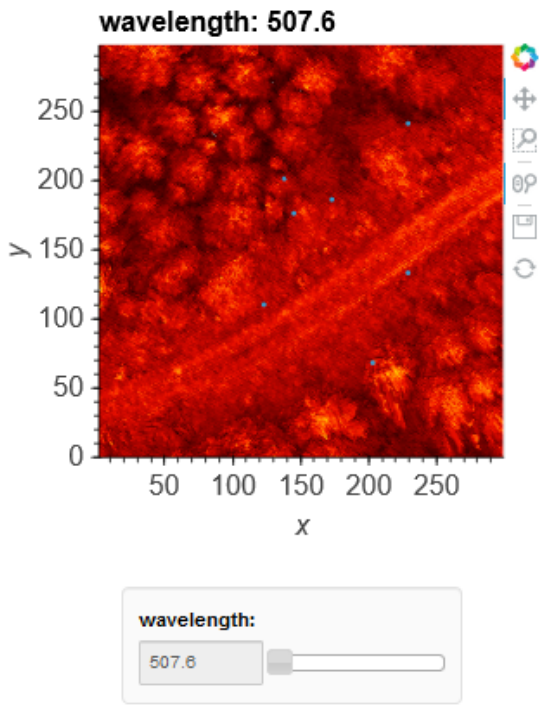

Figure 14. Screenshot of the point chooser.

Finally there is a histogram method (figure 17), that calculates histograms for each bands and shows those histograms side by side. This is translated from hsicube (Eskelinen, 2017) MATLAB package to Python.

2.4 Machine learning

Machine learning can be handled using scikit-learn[6] package (Pedregosa et al., 2011). The main idea of scikit-learn is to make

[6] Scikit-learn can be installed with pip (`pip install sklearn`) or conda (`conda install sklearn`) Python package managers.

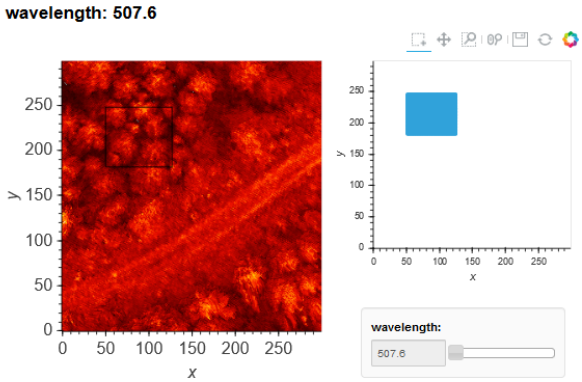

Figure 15. Screenshot of the box chooser.

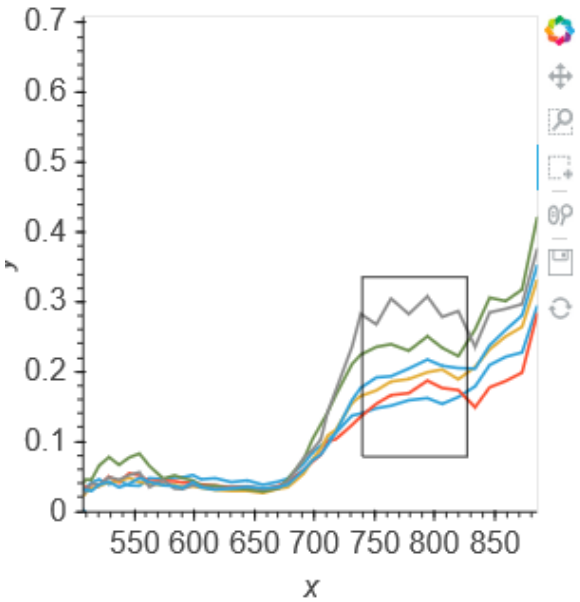

Figure 16. Screenshot of the spectre chooser.

simple and efficient tools for data analysis. Part of the simplicity is documentation, and with scikit-learn it is done well. There is flowchart for finding a suitable estimator, and every estimator is documented so well, that one can easily learn to use them decently.

Another thing we want to point out is the variety of implemented algorithms. Every well established machine learning algorithm can be found. Still, there are no duplicates, and the user does not have to worry about competing implementations, and the API is consistent through the algorithms.

Other useful properties are flows, parallel computing, fine tuning. The user can relatively easily make a workflow, that preprosesses data, does cross-validation on desired estimators with desired parameters and returns the estimator, that seems to produce the best result. The basic forms of the estimators are simple, but there are multiple parameters that one can use to fine tune the estimator.

The usage is simple since the algorithms are well documented and their API is simple, yet agile. Scikit-learn is also free and open source, and it is developed by scikit-learn team. It fullfills

```
result = cube.visualize.histogram(band_dim = 'band')
hist_image = result[0]
hist_counts = result[1]
bin_edges = result[2]
hist_image
```

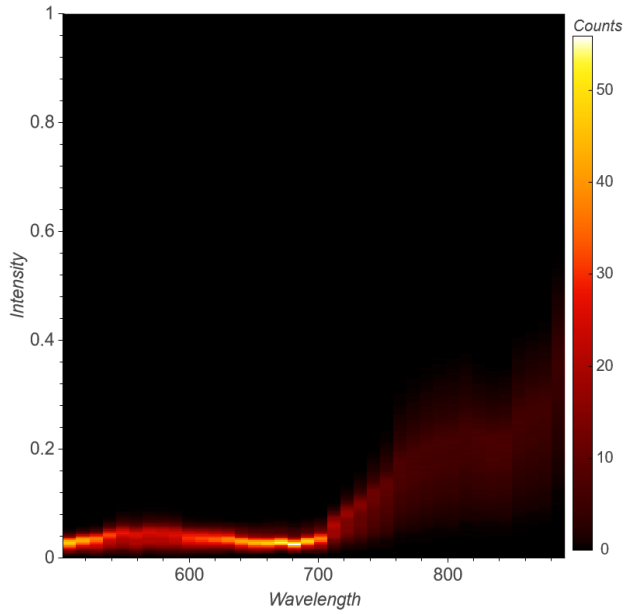

Figure 17. Visualisation of the histogram. The histogram tool returns an image of the histogram, the values of the histogram and the bin edges.

our criteria of being easy to use. It is agile in a way that user can make own flows through the algorithms and the user can fine-tune the algorithms as much as is needed.

Now we can use machine learning on our example problem. First, in figure 18 we use pandas[7] (McKinney, 2010) for reading the tree dataset and plot it over one of the bands in our cube.

Now we can use the tree coordinates to train a machine learning algorithm to recognise birch from pine. We take 30 * 30 box around every tree and calculate histogram of the box. These histograms are used to train the algorithm. We also have to make nan-values zero for this. In figure 19 we use VisualisorAccessor to make the histograms and goal vector and prepare them for machine learning.

In figure 20 we train the machine learning algorithm. For this example we are using support vector machine algorithm. We also do cross-validiation with GridSearchCV. Both of these functions are functions from scikit-learn. Then we print out the results, and that tells us the best accuracy score[8] and the best parameters.

In figure 21 use the predictor to predict the species of a 30x30 histogram, that is made from a hyperspectral image of a tree. From the result we could then interpret wheter the estimator estimates the histogram as a pine or a birch.

[7]Pandas is in practice the Python standard for tabular and Excel type data.

[8]Note, that the score should not be taken too seriously, since this is a toy example, and birch and pine are really easy to recognise from eatch other.

```
trees_panda = pd.read_csv('trees_panda_kuva_2',
                          index_col=0)
trees_panda = trees_panda.sort_values("I"). \
drop_duplicates(["I", "P"],
keep="first")
points = hv.Points(trees_panda,
kdims=['I','P'])
hv.Image(cube[20]) * points
```

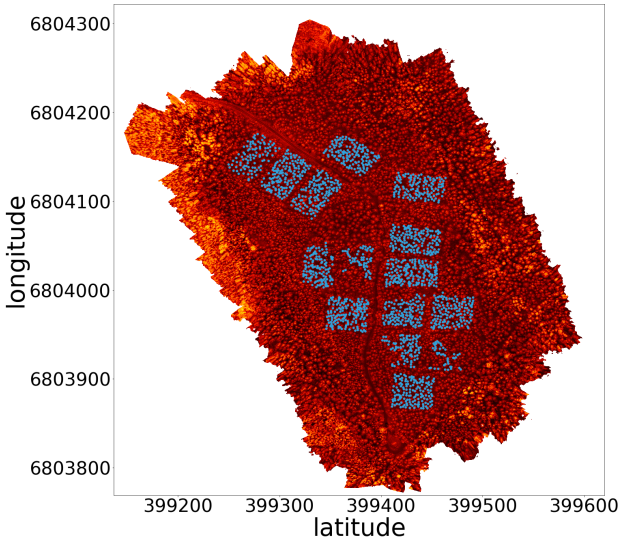

Figure 18. Visualisation of the trees over the image. In this block we read tree data as pandas DataFrame and visualise the trees on the top of one band of the cube.

```
cube.values = np.nan_to_num(cube.values)
X_list = []
size = 30
cellsize = float(cube.attrs['cellsize'])
add = cellsize * size / 2
bin_edges = np.arange(0, 1,  1/20)
i = 0
for puu in trees_panda.values:
print(i)
i = i+1
x_coord = puu[1]
#print(x_coord)
y_coord = puu[2]
#print(y_coord)
cropped = cube.sel(y=slice(y_coord + add,
                           y_coord - add),
                   x=slice(x_coord-add,
                           x_coord+add))
_, hist, _ = cropped.visualize.\
                 histogram(
            bin_edges=bin_edges,
                      flag='linear',
                      show_plot=False,
                      band_dim="wavelength"
            )
hist_flat = np.array(hist).flatten()
X_list.append(hist_flat)
X = np.array(X_list)
y = np.array(trees_panda['pl'])
```

Figure 19. Calculating histograms and preparing data for machine learning algorithm.

2.5 Other aspects

Other notable libraries for hyperspectral data analysis are already mentioned Bokeh for advanced visualisation, scikit-image (van der Walt et al., 2014) for image data analysis and TensorFlow (Abadi et al., 2015) and Keras[9] (Chollet et al., 2015) for deep learning.

```
svc = svm.SVC()
parameters = {'kernel':['linear',
                        'poly',
                        'rbf',
                        'sigmoid'],
              'C':[10**i for i in range(-5,4)]}

clf = GridSearchCV(svc, parameters, n_jobs=20)
clf.fit(X,y)
clf_best = clf.best_estimator_
print(clf.best_score_)
print(clf.best_estimator_)

0.965723612622
SVC(C=0.1, cache_size=200, class_weight=None,
coef0=0.0, decision_function_shape='ovr',
degree=3, gamma='auto', kernel='poly',
max_iter=-1, probability=False,
random_state=None, shrinking=True,
tol=0.001, verbose=False)
```

Figure 20. Here we train the machine learning algorithm and print out the result. caption=Results of the training. Here we see that best estimator predicts correctly 96.6% of the time.

```
tree_1_pred = clf_best.predict(X_new.reshape(1, -1))
```

Figure 21. Here we use the estimator.

Bokeh is a package that has been on the rise in 2017. Bokeh makes interactive Python visualisations, using JavaScript. It is a backend of Holoviews, and if one wants to understand Holoviews deeply, this is one place to look at. Bokeh visualisations are generally quite beautiful, but it comes with expence of computational complexity and increased memory usage.

Scikit-image is a sister package of scikit-learn. Scikit-image is focused on computer vision and image processing. The same advantages as with scikit-learn apply here. The API is consistent and simple and the wide variety of algorithms is well curated.

TensorFlow and Keras are deep learning libraries. TensorFlow is considered to be the state of the art at this field, but the syntax is difficult and learning curve extremely steep. Keras uses TensorFlow as a backend, and offers simpler syntax. If one is a beginner on deep learning, Keras is a library to more easily get started, but as one is becoming more advanced user, TensorFlow's flexibility and increased tuning possibilities start becoming more attractive.

3. CONCLUSIONS AND FURTHER WORK

We have gathered and further developed an agile and easy to use pipeline for hyperspectral data analysis in Python. The tools we have investigated have wide range of advantages such as simple API:s, variety of different implementations, back ends and tools and extendibility.

In addition to that, Python programming language has large user base and active developer community, which guarantees that Python keeps up with needs of scientists. The packages mentioned in this paper are all actively developed and thoroughly tested.

[9] These tools can also be installed with pip or conda.

Also, especially xarray and Holoviews are good Python tools for hyperspectral data processing and visualisation. These tools seem like they are made for this use, but they still provide the generality of non-specialised tools. Compared to HyperSpy end Spectral Python our solution is much more modular and open to extending with new blocks.

Finally we would like to suggest, that in the context of using Python in hyperspectral data analysis, there is need for developing a graphical user interface that uses these tools and finding out best practises for utilising deep learning algorithms. We are starting to develop the graphical user interface in this summer.

Deep learning algorithms are becoming more and more attractive when there is more and more computational power available. The algorithms are computationally intense, but when they are used correctly they provide strikingly good results. These algorithms can be applied on many of the problems on the field of hyperspectral data analysis, such as object recognition, classification or for example analysing the health of a crop.

On this specific toolset there is work to do with parallelisation, since the datasets are huge and paralellisation would make the computations faster.

We have also started to develop a Python library for spectral indices, and are quite far in it already. The leading principle of our implementation is to make a simple implementation of every index on website indexdatabase.de, and wrap the implementation lightly with features that help in the usage. The point was to make the indices easily computable, so that the user could easily use a loop to go through the indices.

One thing we need to define was the API for selecting bands. For many of the indices, they are not defined for exact wavelengths like 745nm, but rather for red light and user needs to define this as he/she wishes. This is for now done by declaring the defaults in form of a Python dictionary. The other thing to consider is how is a band selected. Is it selected only if there is a clear match, the indice wants wavelength 500nm and our data has exactly that or is there room for approximation? If the used data is in format of xarray DataArray or Dataset, then it is possible to use the xarrays nearest neighbor -selection like in figure 5, otherwise one needs to implement their own selector. Once the index library is initialised like in figure 22, one can loop through the indices and find all the indices that can be computed on the dataset like in figure 23. Then the user can plot all possible indices with Holoviews like in figure 24.

```
from pyspindl import Indices, selectors
defaults = {'NIR': 815.7,
            'GREEN': 544.2,
            'RED': 595.3}
defaults.update(
        {k: defaults['RED'] for k in ['Red', 'R']}
        )
defaults['G'] = defaults['GREEN']
#Without defaults, we can not calculate some indices.
indices = Indices(selectors.from_xarray(
                  'wavelength',
                  method='nearest',
                  tolerance=8.0
                  ),
                  defaults=defaults)
```

Figure 22. The initialisation process of spectral indices library. This is still work in progress.

Other thing we we are considering in developing this package is bands. How are they defined? There is big difference between a camera that has the same response on a interval around the middle value and camera that has more gaussian response. These differences should somehow be accounted for with software. The response function could be used in selection, and inbetween values coud be interpolated from two or more bands based on their responses. The response function is definately important in presicion applications and this problem needs to be solved.

```
matches = dict()
for iname, ifunc in indices.items():
    try:
        matches[iname] = ifunc(cube_cropped)
        # The following is necessary to
        # remove indices that result
        # only in +inf, -inf and NaN
        if not np.any(np.isfinite(matches[iname])):
            matches.pop(iname)
            continue
        # We have now built a dictionary
        # of index names and corresponding data.
        matches[iname].coords['index_name'] = iname
        # We also want to clean up
        # unnecessary coordinates, if any remaim
        for coordinate in ['band',
                           'fwhm',
                           'wavelength']:
            if coordinate in matches[iname].coords:
                matches[iname] = matches[iname]. \
                                 drop(coordinate)
    except (KeyError, TypeError, NameError):
        pass
print(str(len(matches))+' matching indices found.')
```

Figure 23. We loop through the indices, and take those that are sensible.

```
%%output size = 250
%%opts Image [invert_yaxis=True] (cmap='Spectral')
dataset = hv.Dataset(prettyfield,
                     kdims=['index_name', 'x', 'y'],
                     vdims='Index')
dataset.to(hv.Image,
           kdims=['x', 'y'],
           dynamic=True).hist()
```

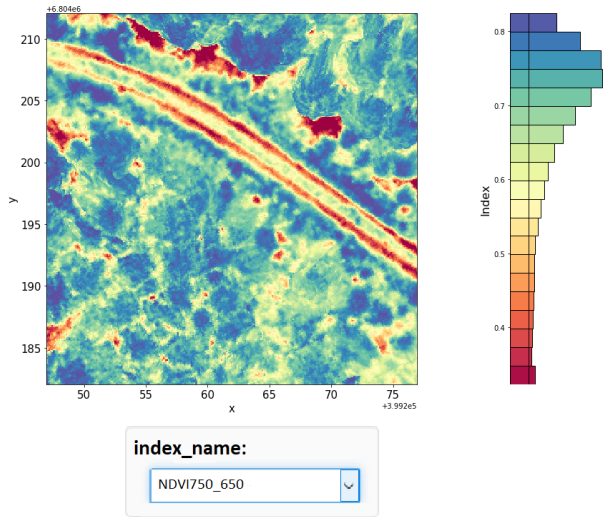

Figure 24. All indices in a dropdown menu. Dropdown menu comes from the use of Holoviews Dataset.

REFERENCES

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S.,

Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X., 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Official website: `https://www.tensorflow.org/`.

Boggs, T., n.d. Spectral python. Source code available at `https://github.com/spectralpython`.

Bokeh Development Team, 2014. Bokeh: Python library for interactive visualization. Official website: `http://www.bokeh.pydata.org`.

Chollet, F. et al., 2015. Keras. Source code available at `https://github.com/keras-team/keras`.

Dask Development Team, 2016. Dask: Library for dynamic task scheduling. Official website: `http://dask.pydata.org`.

de la Peña, F. et al., 2017. hyperspy/hyperspy: Hyperspy 1.3.

Eskelinen, M. A., 2017. Software framework for hyperspectral data exploration and processing in matlab. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-3/W3, pp. 47–50. Source code available at `https://github.com/silmae/hsicube`.

GDAL Development Team, 2018. Gdal - geospatial data abstraction library, version 2.2.3. Official website: `http://www.gdal.org`.

Gillies, S. et al., 2013–. Rasterio: geospatial raster i/o for Python programmers. Source code available at `https://github.com/mapbox/rasterio`.

Hoyer, S. and Hamman, J., 2017. xarray: N-D labeled arrays and datasets in Python. *Journal of Open Research Software*. Source code available at `https://github.com/pydata/xarray`.

Hunter, J. D., 2007. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering* 9(3), pp. 90–95. Source code is available at `https://github.com/matplotlib/matplotlib`.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S. and Willing, C., 2016. Jupyter notebooks – a publishing format for reproducible computational workflows. pp. 87 – 90.

McKinney, W., 2010. Data structures for statistical computing in python. In: S. van der Walt and J. Millman (eds), *Proceedings of the 9th Python in Science Conference*, pp. 51 – 56.

Oliphant, T., 2006. A guide to numpy. Source code available at `https://github.com/numpy/numpy`.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E., 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12, pp. 2825–2830.

Plotly Technologies Inc., 2015. Collaborative data science. Official website: `https://plot.ly`.

Rew, R. K., Davis, G. P., Emmerson, S. and Davies, H., 1997. NetCDF User's Guide for C, An Interface for Data Access, Version 3.

Stevens, J.-L., Rudiger, P. and Bednar, J. A., n.d. Holoviews. Source code available at `https://github.com/ioam/holoviews`.

van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., Yu, T. and the scikit-image contributors, 2014. scikit-image: image processing in Python. *PeerJ* 2, pp. e453. Source code available at `https://github.com/scikit-image/scikit-image`.