hypers Documentation

Release 0.0.12

Priyank Shah

Mar 07, 2021

CONTENTS:

Provides a data structure model for hyperspectral data.

- Simple tools for exploratory analysis of hyperspectral data
- Interactive hyperspectral viewer built into the object
- Allows for unsupervised machine learning directly on the object (using scikit-learn)
- More features coming soon...

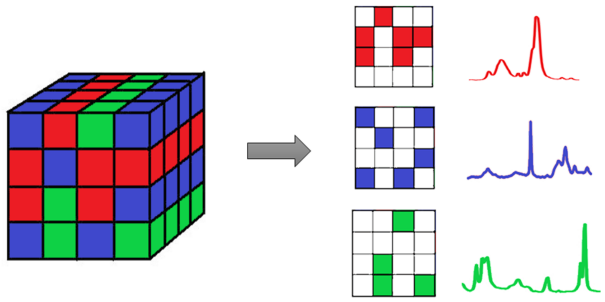

Fig. 1: Extracting class components from hyperspectral data.

CHAPTER

ONE

INSTALLATION

To install using `pip`:

```
pip install hypers
```

1.1 Dependencies

The following packages are required and will be installed when installing `hypers`.

- numpy
- scipy
- matplotlib
- scikit-learn
- PyQt5
- pyqtgraph

CHAPTER

TWO

HPARRAY: AN INTRODUCTION

2.1 Motivation

The motivation behind the creation of this package was performing common tasks on a numpy `ndarray` for hyperspectral data which could be better served by extending the `ndarray` type with added functionality for hyperspectral data. This package provides just that, a *hparray* type that subclasses `ndarray` and adds further functionality. An advantage over other packages is that it the *hparray* object can still be used as a normal numpy array for other tasks.

2.2 Processing data

The hyperspectral data is stored and processed using *hparray*.

Note: Note that the array should be formatted in the following order:

(spatial, spectral)

i.e. the spatial dimensions should proceed the spectral dimension/channels. As an example, if our hyperspectral dataset has dimensions of *x=10*, *y=10*, *z=10* and *channels=100* then the array should be formatted as:

(10, 10, 10, 100)

Below is an example of instantiating a *hparray* object with a 4d random numpy array.

```
import numpy as np
import hypers as hp

test_data = np.random.rand(40, 40, 4, 512)
X = hp.array(test_data)
```

2.3 Properties

The *hparray* object has several useful attributes and methods for immediate analysis:

Note: Note that as *hparray* subclasses numpy's `ndarray`, all the usual methods and attributes in a numpy array can also be used here.

```
# Data properties:
X.shape                              # Shape of the hyperspectral array
X.ndim                               # Number of dimensions
X.nfeatures                          # Size of the spectral dimension/channels
X.nsamples                           # Total number of pixels (samples)
X.nspatial                           # Shape of the spatial dimensions

# To access the mean image/spectrum of the dataset:
X.mean_spectrum
X.mean_image

# To access the image/spectrum in a specific pixel/spectral range:
X.spectrum[10:20, 10:20, :, :]       # Returns spectrum within chosen pixel range
X.image[..., 100:200]                # Returns image averaged between spectral bands

# To view and interact with the data:
X.plot(backend='pyqt')                              # Opens a hyperspectral viewer
```

To view the full list of methods and attributes that the Process class contains, see *hparray*.

CHAPTER

THREE

HPARRAY: REFERENCE

Extends functionality of np.ndarray for hyperspectral data

class hypers.core.array.**hparray**(*input_array: Union[list, numpy.ndarray, hypers.core.array.hparray]*)

Extend functionality of a numpy array for hyperspectral data

The usual *numpy.ndarray* attributes and methods are available as well as some additional ones that extend functionality.

Parameters

input_array: Union[list, np.ndarray] The array to convert. This should either be a 2d/3d/4d numpy array (type *np.ndarray*) or list.

Attributes

mean_image: np.ndarray Provides the mean image by averaging across the spectral dimension. e.g. if the shape of the original array is (100, 100, 512), then the image dimension shape is (100, 100) and the spectral dimension shape is (512,). So the mean image will be an array of shape (100, 100).

mean_spectrum: np.ndarray Provides the mean spectrum by averaging across the image dimensions. e.g. if the shape of the original array is (100, 100, 512), then the image dimension shape is (100, 100) and the spectral dimension shape is (512,). So the mean spectrum will be an array of shape (512,).

Methods

all([axis, out, keepdims, where])	Returns True if all elements evaluate to True.
any([axis, out, keepdims, where])	Returns True if any of the elements of *a* evaluate to True.
argmax([axis, out])	Return indices of the maximum values along the given axis.
argmin([axis, out])	Return indices of the minimum values along the given axis.
argpartition(kth[, axis, kind, order])	Returns the indices that would partition this array.
argsort([axis, kind, order])	Returns the indices that would sort this array.
astype(dtype[, order, casting, subok, copy])	Copy of the array, cast to a specified type.
byteswap([inplace])	Swap the bytes of the array elements
choose(choices[, out, mode])	Use an index array to construct a new array from a set of choices.

continues on next page

Table 1 – continued from previous page

clip([min, max, out])	Return an array whose values are limited to [min, max].
collapse()	Collapse the array into a 2d array
compress(condition[, axis, out])	Return selected slices of this array along given axis.
conj()	Complex-conjugate all elements.
conjugate()	Return the complex conjugate, element-wise.
copy([order])	Return a copy of the array.
cumprod([axis, dtype, out])	Return the cumulative product of the elements along the given axis.
cumsum([axis, dtype, out])	Return the cumulative sum of the elements along the given axis.
diagonal([offset, axis1, axis2])	Return specified diagonals.
dot(b[, out])	Dot product of two arrays.
dump(file)	Dump a pickle of the array to the specified file.
dumps()	Returns the pickle of the array as a string.
fill(value)	Fill the array with a scalar value.
flatten([order])	Return a copy of the array collapsed into one dimension.
getfield(dtype[, offset])	Returns a field of the given array as a certain type.
item(*args)	Copy an element of an array to a standard Python scalar and return it.
itemset(*args)	Insert scalar into an array (scalar is cast to array's dtype, if possible)
max([axis, out, keepdims, initial, where])	Return the maximum along a given axis.
mean([axis, dtype, out, keepdims, where])	Returns the average of the array elements along given axis.
min([axis, out, keepdims, initial, where])	Return the minimum along a given axis.
newbyteorder([new_order])	Return the array with the same data viewed with a different byte order.
nonzero()	Return the indices of the elements that are non-zero.
partition(kth[, axis, kind, order])	Rearranges the elements in the array in such a way that the value of the element in kth position is in the position it would be in a sorted array.
plot([backend])	Interactive plotting to interact with hyperspectral data
prod([axis, dtype, out, keepdims, initial, ...])	Return the product of the array elements over the given axis
ptp([axis, out, keepdims])	Peak to peak (maximum - minimum) value along a given axis.
put(indices, values[, mode])	Set a.flat[n] = values[n] for all *n* in indices.
ravel([order])	Return a flattened array.
repeat(repeats[, axis])	Repeat elements of an array.
reshape(shape[, order])	Returns an array containing the same data with a new shape.
resize(new_shape[, refcheck])	Change shape and size of array in-place.
round([decimals, out])	Return *a* with each element rounded to the given number of decimals.
searchsorted(v[, side, sorter])	Find indices where elements of v should be inserted in a to maintain order.

continues on next page

Table 1 – continued from previous page

setfield(val, dtype[, offset])	Put a value into a specified place in a field defined by a data-type.
setflags([write, align, uic])	Set array flags WRITEABLE, ALIGNED, (WRITEBACKIFCOPY and UPDATEIFCOPY), respectively.
smoothen([method])	Returns smoothened hp.hparray
sort([axis, kind, order])	Sort an array in-place.
squeeze([axis])	Remove axes of length one from *a*.
std([axis, dtype, out, ddof, keepdims, where])	Returns the standard deviation of the array elements along given axis.
sum([axis, dtype, out, keepdims, initial, where])	Return the sum of the array elements over the given axis.
swapaxes(axis1, axis2)	Return a view of the array with *axis1* and *axis2* interchanged.
take(indices[, axis, out, mode])	Return an array formed from the elements of *a* at the given indices.
tobytes([order])	Construct Python bytes containing the raw data bytes in the array.
tofile(fid[, sep, format])	Write array to a file as text or binary (default).
tolist()	Return the array as an `a.ndim`-levels deep nested list of Python scalars.
tostring([order])	A compatibility alias for *tobytes*, with exactly the same behavior.
trace([offset, axis1, axis2, dtype, out])	Return the sum along diagonals of the array.
transpose(*axes)	Returns a view of the array with axes transposed.
var([axis, dtype, out, ddof, keepdims, where])	Returns the variance of the array elements, along given axis.
view([dtype][, type])	New view of array with the same data.

collapse()

Collapse the array into a 2d array

Collapses the array into a 2d array, where the first dimension is the collapsed image dimensions and the second dimension is the spectral dimension.

Returns

np.ndarray The collapsed 2d numpy array.

Examples

```
>>> import numpy as np
>>> import hypers as hp
>>> data = np.random.rand(40, 30, 1000)
>>> x = hp.hparray(data)
>>> collapsed = x.collapse()
>>> collapsed.shape
(1200, 1000)
```

Return type ndarray

property nfeatures

Returns the number of features (size of the spectral dimension) in the dataset

Returns

int: Size of the spectral dimension

property nsamples
Returns the number of samples (total number of spatial pixels) in the dataset

Returns

int: Total number of samples

property nspatial
Returns the shape of the spatial dimensions

Returns

tuple: Tuple of the shape of the spatial dimensions

plot(*backend='pyqt'*)
Interactive plotting to interact with hyperspectral data

Note that at the moment only the 'pyqt' backend has been implemented. This means that PyQt is required to be installed and when this method is called, a separate window generated by PyQt will pop up. It is still possible to use this in a Jupyter environment, however the cell that calls this method will remain frozen until the window is closed.

Parameters

backend: str Backend to use. Default is 'pyqt'.

smoothen(*method='savgol'*, ***kwargs*)
Returns smoothened hp.hparray

Parameters

method: str Method to use to smooth the array. Default is 'savgol'. + 'savgol': Savitzky-Golay filter.

****kwargs** Keyword arguments for the relevant method used. + method='savgol'

kwargs for the *scipy.signal.savgol_filter* implementation

Returns

hp.hparray The smoothened array with the same dimensions as the original array.

Return type *hparray*

CHAPTER

FOUR

UNSUPERVISED LEARNING

The `hparray` object has built in methods that allows you to perform several unsupervised learning techniques on the stored data. The techniques are split into the following categories:

- Spectral unmixing
- Abundance mapping

These are all available as methods on the `hparray` object.

```
import numpy as np
import hypers as hp

test_data = np.random.rand(10, 10, 1000)
X = hp.array(test_data)

# To access vertex component analysis
ims, spcs = X.unmix.vca.calculate(n_components=10)

# To access unconstrained least-squares for abundance mapping
spectra = np.random.rand(1000, 2)
amap = X.abundance.ucls.calculate(spectra)
```

4.1 Spectral unmixing

Spectral unmixing is the process of decomposing the spectral signature of a mixed pixel into a set of endmembers and their corresponding abundances.

The following techniques are available:

- Vertex component analysis

4.1.1 Vertex component analysis

This is implemented with[1].

class hypers.learning.decomposition.**vca**(*X*)

[1] VCA algorithm. J. M. P. Nascimento and J. M. B. Dias, "Vertex component analysis: a fast algorithm to unmix hyperspectral data," in IEEE Transactions on Geoscience and Remote Sensing, 2005. Adapted from repo.

Methods

calculate	

4.2 Abundance mapping

Abundance maps are used to determine how much of a given spectrum is present at each pixel in a hyperspectral image. They can be useful for determining percentages after the spectra have been retrieved from some clustering or unmixing technique or if the spectra are already at hand.

The following techniques are available:

- Unconstrained least-squares
- Non-negative constrained least-squares
- Fully-constrained least-squares

4.2.1 Unconstrained least-squares

This is implemented with[2].

class hypers.learning.abundance.**ucls**(*X*)

Methods

calculate	

4.2.2 Non-negative constrained least-squares

This is implemented with[2].

class hypers.learning.abundance.**nnls**(*X*)

Methods

calculate	

[2] Abundance mapping. Adapted from PySptools.

4.2.3 Fully-constrained least-squares

This is implemented with[Page 12, 2].

class hypers.learning.abundance.**fcls**(X)

Methods

calculate	

References

CHAPTER

FIVE

HYPERSPECTRAL DATA VIEWER

Included with *hypers* is a hyperspectral data viewer that allows for visualization and interactivity with the hyperspectral dataset.

- From the `Dataset` instance variable:

```
import numpy as np
import hypers as hp

test_data = np.random.rand(100, 100, 5, 512)
X = hp.Dataset(test_data)

X.view()
```

The hyperspectral data viewer is a lightweight pyqt gui. Below is an example:

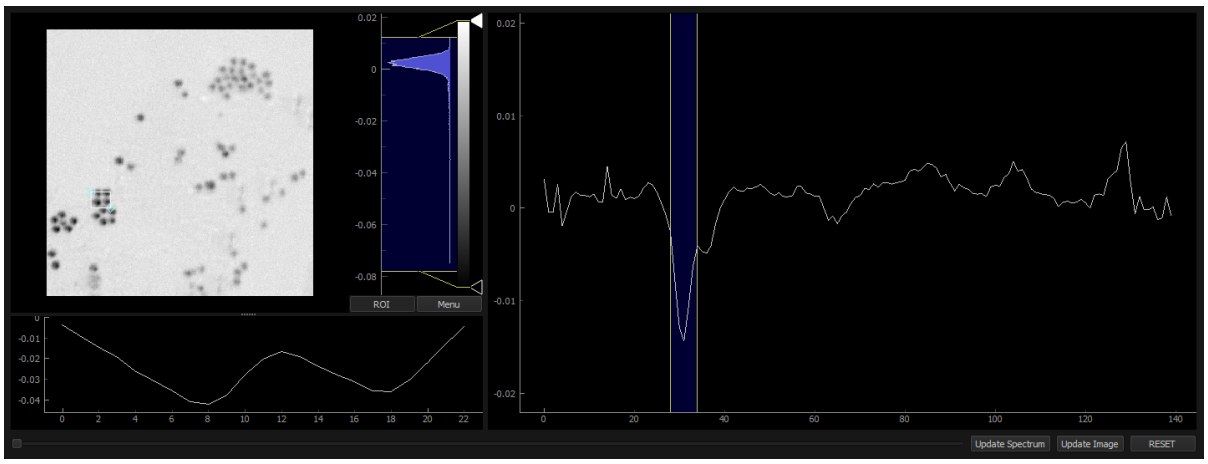

Fig. 1: Hyperspectral data viewer.

Note: If using `hypers` in a Jupyter notebook, it is still possible to use the data viewer. However the notebook cell will be frozen until the data viewer has been closed.

This is due to the fact that the data viewer uses the same CPU process as the notebook. This may be changed in the future.

PYTHON MODULE INDEX

h

INDEX