hypers Documentation

Release 0.0.10

Priyank Shah

Jan 02, 2019

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. . .

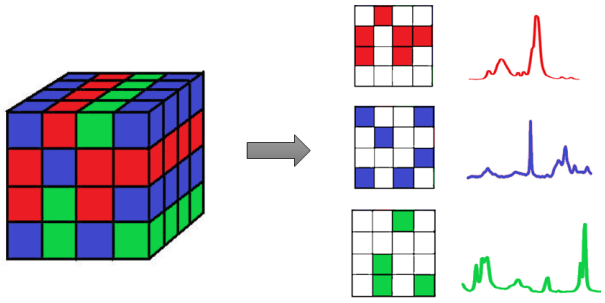

Fig. 1: Extracting class components from hyperspectral data.

CHAPTER 1

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 2

Dataset: An introduction

2.1 Processing data

The hyperspectral data is stored and processed using `Dataset`.

Note: Note that the Dataset class will only accept a `numpy` array of dimensions 3 or 4. The array should be formatted as:

(x, y, spectrum) or *(x, y, z, spectrum)*

Below is an example of instantiating a `Dataset` 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.Dataset(test_data)
```

2.2 Dataset properties

The `Dataset` object has several useful attributes and methods for immediate analysis:

```
# Data properties:
X.shape                             # Shape of the hyperspectral array
X.ndim                              # Number of dimensions (3 or 4)
X.n_features                        # Number of spectral points (features)
X.n_samples                         # Total number of pixels (samples)

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

(continues on next page)

(continued from previous page)

```
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 access the scree plot (as an array) that explains the variance contribution:
X.scree()

# To view and interact with the data:
X.view()                            # Opens a hyperspectral viewer
```

The `Dataset` object also supports arithmetic operations in the following manner:

```
import numpy as np
import hypers as hp

test_data = np.random.rand(50, 50, 512)
spectral_array = np.random.rand(512)

X = hp.Dataset(test_data)

# For arithmetic operations with a constant (int or float)
# This will be performed element-wise (on every single spectral band at every single␣
↪pixel)
X *= 2
X /= 2
X += 2
X -= 2

# For arithmetic operations with a spectrum (spectral_array must have the same size␣
↪as the spectra in Dataset)
# This will be performed on every single spectrum at every pixel
X *= spectral_array
X /= spectral_array
X += spectral_array
X -= spectral_array
```

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

CHAPTER 3

Dataset: Reference

Stores data in a custom class and generates attributes for other modules

class hypers._process.**Dataset**(*data*)

Dataset data structure

Attributes

data: np.ndarray The raw hyperspectral data

shape: tuple Shape of the hyperspectral data

ndim: int Number of dimensions of the data (3 or 4)

n_features: int Number of spectral bands

n_samples: int Total number of pixels

mean_spectrum: np.ndarray An array of the mean spectrum of the data

mean_image: np.ndarray An array of the mean image of the data

image: np.ndarray An array of the image averaged over the specified spectral bands

spectrum: np.ndarray An array of the spectrum averaged over the specified pixels

smoothing: str Smoothing to use. Can be either:

- savitzky_golay
- gaussian

Default is 'savitzky_golay'.

Methods

abundance(spectra[, plot, return_arrs])	Abundance map with least-squares fitting

Continued on next page

Table 1 – continued from previous page

cluster(mdl[, decomposed, pca_comps, plot, ...])	Cluster data
decompose(mdl[, plot, return_arrs])	Decompose data
flatten()	Returns flattened array
mixture(mdl[, plot, return_arrs])	Gaussian mixture models
normalize([norm])	Normalize the hyperspectral data
preprocess(mdl)	Preprocess data
robust_scale([with_centering, with_scaling, ...])	Standardize the hyperspectral data (robust from outliers)
scale([with_mean, with_std])	Standardize the hyperspectral data
scree([plot, return_arrs])	Returns PCA scree
smoothen(**kwargs)	Smoothen the hyperspectral data
update()	Update the stored data and class properties
vca([n_components, plot, return_arrs])	Vertex component analysis
view()	Open the hyperspectral viewer GUI

abundance(*spectra*, *plot=False*, *return_arrs=True*)
Abundance map with least-squares fitting

Parameters

spectra: np.ndarray Spectra to perform fitting with

plot: bool If True, will return an image of the abundance map

return_arrs: bool If True, will return an array of the abundance map

Returns

im: np.ndarray An array of the abundance map

Return type `ndarray`

cluster(*mdl*, *decomposed=False*, *pca_comps=4*, *plot=False*, *return_arrs=True*)
Cluster data

Parameters

mdl: ClusterType Accepts a `scikit-learn` cluster class

decomposed: bool Whether to perform PCA on the data prior to clustering

pca_comps: int If decomposed=True, this specifies the number of principal components to use

plot: bool If True, will return a plot of the labels/spectra of the clusters

return_arrs: bool If True, will return the arrays of the labels/spectra of the clusters

Returns

ims: np.ndarray An array of the labels (size: x, y, (z))

specs: np.ndarray An array of spectra (size: spectra, n_clusters)

Return type `Tuple[ndarray, ndarray]`

decompose(*mdl*, *plot=False*, *return_arrs=True*)
Decompose data

Parameters

mdl: DecomposeType Accepts a `scikit-learn` decomposition class

plot: bool If True, will return a plot of the images/spectra of the components

return_arrs: bool If True, will return the arrays of the images/spectra

Returns

ims: np.ndarray An array of images (size: x, y, (z), n_components)

specs: np.ndarray An array of spectra (size: spectra, n_components)

Return type `Tuple[ndarray, ndarray]`

flatten()
Returns flattened array

Returns

x_flat: np.ndarray Flattened 2d array of the stored hyperspectral data

Return type `ndarray`

mixture(*mdl*, *plot=False*, *return_arrs=True*)
Gaussian mixture models

Parameters

mdl: MixtureType Accepts a `scikit-learn` mixture class

plot: bool If True, will return a plot of the labels/spectra of the components

return_arrs: bool If True, will return the arrays of the labels/spectra of the components

Returns

labels: np.ndarray An array of the labels

spectra: np.ndarray An array of the spectra of the components

Return type `Tuple[ndarray, ndarray]`

normalize(*norm='l2'*)
Normalize the hyperspectral data

Parameters

norm: str ('l1', 'l2', 'max') The norm to use to normalize each spectrum.

Return type None

preprocess(*mdl*)
Preprocess data

Parameters

mdl: PreprocessType Accepts a `scikit-learn` preprocessing class

Return type None

robust_scale(*with_centering=True*, *with_scaling=True*, *quantile_range=(25.0, 75.0)*)
Standardize the hyperspectral data (robust from outliers)

Parameters

with_centering: bool If True, center the data before scaling

with_scaling: bool If True, scale the data to unit variance

quantile_range: tuple Quantile range used.

Return type None

scale(*with_mean=True*, *with_std=True*)
Standardize the hyperspectral data

Parameters

with_mean: bool If True, center the data before scaling

with_std: bool If True, scale the data to unit variance

Return type None

scree(*plot=False*, *return_arrs=True*)
Returns PCA scree

Parameters

plot: bool If True, will plot the array

return_arrs: bool If True, will return the array

Returns

scree: np.ndarray An array of the PCA scree

Return type ndarray

smoothen(***kwargs*)
Smoothen the hyperspectral data

Parameters

****kwargs** Keyword arguments for either of the following (depending on what the smoothing attribute has been set to):

- scipy.signal.savgol_filter
- scipy.ndimage.filters.gaussian_filter

Return type None

update()
Update the stored data and class properties

Return type None

vca(*n_components=4*, *plot=False*, *return_arrs=True*)
Vertex component analysis

Parameters

n_components: int Number of pure components to find

plot: bool If True, will return a plot of the pure spectra

return_arrs: bool If True, will return an array of the pure spectra and a list of tuples of the coordinates of the pure pixels

Returns

spectra: np.ndarray An array of the spectra of the pure pixels

coords: np.ndarray A list of tuples of the coordinates of the pure pixels

Return type `Tuple[ndarray, List[int]]`

view()

Open the hyperspectral viewer GUI

Return type None

CHAPTER 4

Preprocessing

Preprocessing data with `hypers` is performed directly on the Dataset object. At the moment, the following preprocessing classes from `scikit-learn` are supported:

- MaxAbsScaler
- MinMaxScaler
- PowerTransformer
- QuantileTransformer
- RobustScaler
- StandardScaler

```
import numpy as np
import hypers as hp
from sklearn.preprocessing import StandardScaler

tst_data = np.random.rand(50, 50, 100)
X = hp.Dataset(tst_data)

X.preprocess(
    mdl=StandardScaler()
)
```

CHAPTER 5

Dimensionality Reduction

Dimensionality reduction with `hypers` is performed directly on the Dataset object. At the moment, the following classes from `scikit-learn` are supported:

- PCA
- IncrementalPCA
- TruncatedSVD
- FastICA
- DictionaryLearning
- MiniBatchDictionaryLearning
- FactorAnalysis
- NMF
- LatentDirichletAllocation

```
import numpy as np
import hypers as hp
from sklearn.decomposition import PCA

tst_data = np.random.rand(50, 50, 1000)
X = hp.Dataset(tst_data)

# Retrieving images and spectra of the first 10 principal components of the dataset
ims, spcs = X.decompose(
    mdl=PCA(n_components=10)
)
```

CHAPTER 6

Clustering

Clustering data with `hypers` is performed directly on the Dataset object. At the moment, the following clustering classes from `scikit-learn` are supported:

- KMeans
- AgglomerativeClustering
- SpectralClustering

Clustering can be performed on both the data stored in the Process object itself, or on a set of principal components of the dataset (as demonstrated below).

```
import numpy as np
import hypers as hp
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

tst_data = np.random.rand(50, 50, 1000)
X = hp.Dataset(tst_data)

# Clustering on the stored data first
lbls_nodecompose, spcs_nodecompose = X.cluster(
    mdl=KMeans(n_clusters=4),
    decomposed=False
)

# Clustering on the first 5 principal components on the dataset
lbls_decomposed, spcs_decomposed = X.cluster(
    mdl=KMeans(n_clusters=5),
    decomposed=True,
    pca_comps=5
)
```

CHAPTER 7

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:

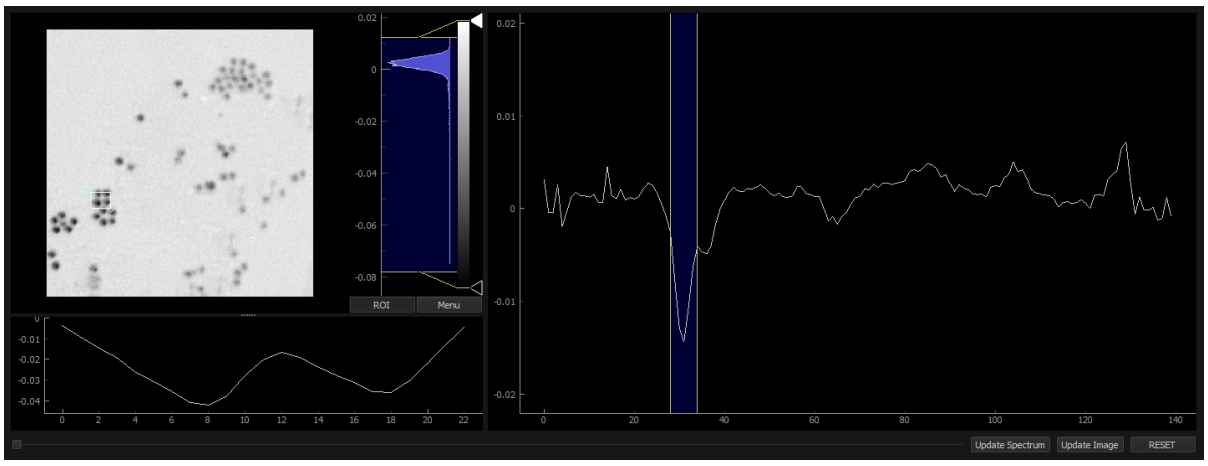

Fig. 1: Hyperspectral data viewer.

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