geospatial-learn Documentation

Release 0.3

Ciaran Robb

Nov 15, 2019

Contents

geospatial-learn is a Python module for using scikit-learn and xgb models with geo-spatial data, chiefly raster and vector formats.

The module also contains various fuctionality for manipulating raster and vector data as well as some utilities aimed at processing Sentinel 2 data.

The aim is to produce convenient, minimal commands for putting together geo-spatial processing chains using machine learning libs. Development will aim to expand the variety of libs/algorithms available for machine learning beyond the current complement.

CHAPTER 1

Contents

1.1 Quickstart

1.1.1 Notes

Be sure to replace the paths with paths to your own imagery/polygons!

1.1.2 Training and model creation

The following simple example uses the learning module to read in training from a shapefile and associated raster, then exhaustively grid search the model based on a default range of parameters. It is also possible to pass sklearn parameteter dicts to the create_model function.

Bear in mind a large amount of training data and a lot of paramter combinations results in many model fits and lengthy grid search time!

```
# Import the module required
from geospatial_learn import learning

# collect some training data
trainShape = 'path/to/my/trainingShp.shp'
inRas = 'path/to/my/rasterFile.shp'

# training collection, returning any rejects (invalid geometry - rej)
# the 'Class' string is the title of the training label field attribute
training, rej = learning.get_training(trainShape, inRas, 8, 'Class')

# path to my model
model = 'path/to/my/model.gz'


#
```

(continues on next page)

(continued from previous page)

```
results = learning.create_model(training, model, clf='rf', cv=3,
                                cores = 8, strat=True)
```

1.1.3 Classification

The following code uses the learning module to classify an image based on the model made in the code above.

```
from geospatial_learn import learning

# no of bands in raster
bands = 8

# path to output map
outMap = 'path/to/my/rasterFile'

learning.classify_pixel_bloc(model, inRas, bands, outMap,  blocksize=256)
```

1.1.4 Polygon processing

Add attributes to a shapefile - perhaps with a view to classifying them later.

The following calculates some geometric properties and pixel based statistics using functions from the shape module.

```
from geospatial_learn.shape import shape_props, zonal_stats

# path to polygon
segShp = 'path/to/my/segmentShp.shp'

# function to write

# Property of interest
prop = 'Eccentricity'

# function
shape_props(segShp, prop, inRas=None,  label_field='ID')

# variables for function
band = 1
inRas = 'pth/to/myraster.tif'
bandname = 'Blue'

# function
zonal_stats(segShp, inRas, band, bandname, stat = 'mean',
             write_stat=True, nodata_value=None)
```

To write multiple attributes a simple loop will suffice:

```
# shape props
sProps = ['MajorAxisLength', 'Solidity']

for prop in sProps:
   shape_props(segShp, prop, inRas=None,  label_field='ID')
```

(continues on next page)

(continued from previous page)

```
# zonal stats
# please note that by using enumerate we assume the bandnames are ordered as the are␣
↪in the image!
bandnames = ['b', 'g', 'r', 'nir']


# Please note we add 1 to the bnd index as python counts from zero
for bnd,name in enumerate(bandnames):
   zonal_stats(segShp, inRas, bnd+1, name, stat = 'mean', write_stat = True)
```

1.1.5 Train & then classify shapefile attributes

In the previous example several attributes were calculated and written to a shapefile. The following example outlines how to train a ML model then classify these. In this case the attributes are some of those calculated above

1.1.6 Training

For training a model using shape attributes, an attribute containing the Class label (this can be done manually in any GIS) as well as feature attributes are required. We enter the column index of the Class label attribute. In this example it is column 1.

The remaining attributes are assumed to be features (here we are using the ones calculated in the above looped examples).

```
# collect some training data

label_field = 'Class'

feat_fields = ['b', 'g', 'r', 'nir','MajorAxisLength', 'Solidity']

training = path/to/my/training.gz

get_training_shp(inShape, label_field, feat_fields,  outFile = training)
```

The model is created in the same way as the image based method outlined earlier (see Training and model creation). After this the shapefile attributes are classified with the model as shown below and the results are written as a new attribute 'ClassRf'

```
attributes = ['b', 'g', 'r', 'nir','MajorAxisLength', 'Solidity']

classify_object(model, segShp, attributes, field_name='ClassRf')
```

1.1.7 Sentinel 2 data

The following code will stack a set of Sentinel 2 (S2) bands into a single raster. The code uses the module 'geodata', which has a range of functions for manipulating raster data. I have used a genuine S2 path here hence the extreme length of the string!

The function automatically names the stacked raster and saves it in the granule folder.

```
from geospatial_learn import geodata

path = '/path/to/S2A_MSIL1C_20161223T075332_N0204_R135_T36MYE_20161223T080853/S2A_
↪MSIL2A_20161223T075332_N0204_R135_T36MYE_20161223T080853.SAFE/GRANULE/L2A_T36MYE_
↪A007854_20161223T080853/'

outputPth = geodata.stack_S2(path)
```

1.2 geospatial_learn package

1.2.1 Submodules

1.2.2 geospatial_learn.data module

1.2.3 geospatial_learn.geodata module

1.2.4 geospatial_learn.learning module

1.2.5 geospatial_learn.shape module

1.2.6 geospatial_learn.utilities module

Created on Thu Sep 8 22:35:39 2016 @author: Ciaran Robb Research Associate in Earth Observation Centre for Landscape and Climate Research (CLCR) Department of Geography, University of Leicester, University Road, Leicester, LE1 7RH, UK

If you use code to publish work cite/acknowledge me and authors of libs etc as appropriate

utilities.**min_bound_rectangle**(*points*)

Find the smallest bounding rectangle for a set of points. Returns a set of points representing the corners of the bounding box. :Parameters: **points** (*list*) – An nx2 iterable of points

Returns an nx2 list of coordinates

Return type list

1.2.7 Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

u

Index