Even More GDAL

Even More GDAL

Overview

1. Python GDAL Functions Lacking Docs
2. Reprojecting Rasters
3. Polygonizing and Rasterizing
4. Numpy Convenience Functions
5. Other Raster Processing Libraries
    - PIL(low)
    - scikit-image
    - scipy.ndimage
6. Demo of ndimage Generic Filter

Python GDAL Functions Lacking Docs

Python GDAL Functions Lacking Docs

- GDAL functions available in python that are not class methods are not well documented
- One option is to look at the GDAL PDF I linked to
    - This is out of date: some new functions missing while old functions no longer valid
- Can also use `help(gdal)` in the interpreter
    - always up-to-date with your version of GDAL

Reprojecting Rasters

Reprojecting Rasters

- Reprojection is a complex process involving warping and resampling
- To warp, need to use a Virtual Format (VRT) raster
  - `gdal.AutoCreateWarpedVRT(src_ds, src_wkt=None, dst_wkt=None, eResampleAlg=GRA_NearestNeighbor, max_error=0.0)`
  - This function will create a blank warped VRT in memory
- To actually copy the data and resample, use the reproject function
  - `gdal.ReprojectImage(src_ds, dst_name, src_wkt=None, dst_wkt=None, eResampleAlg=GRA_NearestNeighbor, warp_memory=0.0, max_error=0.0)`

Reprojecting Rasters

- GDAL has numerous options for resampling methods:
  - GRA_Average = 5
  - GRA_Bilinear = 1
  - GRA_Cubic = 2
  - GRA_CubicSpline = 3
  - GRA_Lanczos = 4
  - GRA_Mode = 6
  - GRA_NearestNeighbour = 0

Reprojecting Rasters

```
from osgeo import gdal, osr

# open the raster dataset
in_ds = gdal.Open(r"C:\data\rasters\slope.img")

# create a destination CRS
dst_crs = osr.SpatialReference()
dst_crs.ImportFromEPSG(4326)  # WGS84

# need to warp the image: use a VRT
vrt = gdal.AutoCreateWarpedVRT(in_ds,
                               in_ds.GetProjection(),
                               dst_crs.ExportToWkt(),
                               gdal.GRA_Cubic)

# new reproject source to vrt
gdal.ReprojectImage(in_ds, vrt, in_ds.GetProjection(),
                    dst_crs.ExportToWkt(), gdal.GRA_Cubic)

# vrt is in memory, so copy to disk
driver = dataset.GetDriver()
out_ds = driver.CreateCopy(r"C:\data\rasters\slope_wgs84", vrt)

# close everything
out_ds = None
vrt = None  # though exists in memory, this is important
in_ds = None
```

Polygonizing and Rasterizing

Polygonizing and Rasterizing

- GDAL in python also has functions to polygonize and rasterize data
    - `Polygonize(src_band, mask_band, outLayer, PixValField, options=None, callback=0, callback_data=None)`
    - `RasterizeLayer(dataset, bands_list, layer, pfnTransformer=None, pTransformArg=None, burn_values=[0], options=None, callback=0, callback_data=None)`

Numpy Convenience Functions

Numpy Convenience Functions

- Numpy functions are easy because they are vectorized
    - That is, they take an array as the input and automatically iterate through the array for you
- If you have a function that is not a vectorized function, but want to use it with an array, you can vectorize it

```
import numpy

# define a function that takes one value at a time
def difference_bigger(x, y):
    return y - x if y > x else x - y

# vectorize the function to allow it to operate on an array
vector_diff_bigger = numpy.vectorize(difference_bigger)

# create an array to process
array = numpy.random.rand(100,100,100) * 100

# call the vectorized function with on the array
out_array = vector_diff_bigger(array, 32.5)
```

Numpy Convenience Functions

- What if you do not want to vectorize your function to the cell level but to the axis level?
    - `numpy.apply_along_axis(function, axis, array, args=None, kwargs=None)`

```
# create a function to return the band with the max value
def max_band(array):
    max_val = array[0]
    max_index = 1

    for index, item in enumerate(array, 1):
        if item > max_val:
            max_val = item
            max_index = index

    return max_index

# create a random array
array = numpy.random.rand(100,100,100) * 100

# apply function on axis 2 of the array (the "band stack")
out_array = numpy.apply_along_axis(max_band, 2, array)
```

Other Raster Processing Libraries

Other Raster Processing Libraries

PIL(low)

- PIL is the Python Imaging Library
- Pillow is a maintained fork of the PIL
- Not super useful for geospatial image processing as is mainly for image editing
    - Some examples make use of PIL
- Most relevant functionality can be replaced by scikit-image

Other Raster Processing Libraries

scikit-image

- An image processing library closely connected with the scipy stack
- Many useful functions and utilities, including:
  - colorspace conversions
  - test data
  - feature detection
  - various filters
  - least cost routing
  - measurements
  - morphology (useful for creating window footprints)
  - viewers and other visualization tools

Other Raster Processing Libraries

scipy.ndimage

- scipy functions for working with n-dimensional arrays representing images
- scikit-image extends this module, so much is duplicated between the two
- ndimage has the generic filter
    - easy way to implement a moving window analysis!

Demo Generic Filter