Capacitación Python
Laboratorio 4

Nelson R. Salinas

Abril 23, 2018

Existe un gran número de librerías de Python que maximizan su aplicabilidad a campos específicos. En el presente laboratorio realizaremos algunos ejercicios con información geoespacial utilizando las librerías GDAL y Numpy.

1. Librerías

1.1. Numpy

Numpy es la librería numérica por excelencia de Python. La mayoría de las funciones de esta librería están implementadas en un lenguaje de bajo nivel (C y Fortran), por lo cual sus tiempos de ejecución son muy cortos.

1.2. GDAL

La librería GDAL de Python es simplemente una interfaz a la librería GDAL de C++, ampliamente usada para el procesamiento de datos espaciales. Es capaz de leer los principales formatos de archivos raster y vectoriales, y puede convertirlos a matrices de Numpy.

1.3. Instalación

La instalación de ambas librerías es facilitada por el sistema de distribución Conda. Simplemente se ejecuta el comando `conda install gdal` desde la terminal de comando del paquete Anaconda. Dado que la librería Numpy es un prerrequisito de GDAL, ambos paquetes serán instalados con dicho comando.

2. Descripción del problema

Dados unos datos de precipitación y elevación, realizar una estimación del área aproximada del área boscosa para cada una de las zonas de vida para Colombia.

3. Insumos

La información de precipitación y elevación sera consultada a través de dos archivos raster. La información concerniente a la presencia de coberturas de bosque en el país esta disponible a través de la capa bosque-no bosque del 2016 producida por el IDEAM y transformada al sistema de coordenadas WGS84. Los archivos de precipitación y elevación comparten las mismas características de resolución y extensión geográfica. El archivo de bosque-no bosque abarca otros límites y es de mayor resolución.

4. Resumen de la solución propuesta

Tal vez la forma más simple de idealizar una solución es estimando la zona de vida en cada uno de los pixeles de los rasters de precipitación y elevación. Los pasos de esta alternativa serían:

1. Inicializar una estructura de datos para llevar registro de los resultados.
2. Leer los metadatos de los archivos raster.
3. Cargar en memoria los rasters de precipitación y elevación.
4. Realizar una iteración a través de cada pixel de los archivos de precipitación y elevación, y en cada uno realizar los siguientes cómputos:
   a) Estimar la zona de vida.
   b) Adquirir las coordenadas geográficas.
   c) Acceder a los datos del raster bosque-no bosque correspondientes a las coordenadas obtenidas.
   d) Si el pixel está ubicado en una zona de bosque, añadirlo al registro como una ubicación adicional para la respectiva zona de vida.
5. Multiplicar los valores encontrados por el área estimada para cada pixel de precipitación/elevación.

5. Estimación zonas de vida

Para estimar la zona de vida se puede implementar una función que contenga la lógica correspondiente. En el código 1 se presenta una version de dicha función. Digítela en una celda de un cuaderno de Jupyter y pruebe su funcionalidad con valores aleatorios de altura y elevación.

Código 1: Función de estimación de zonas de vida para Colombia.

```
def holdridge(altitude, precipitation):
    out = None
    if altitude < 1000:
        if 500 <= precipitation < 1000:
            out = 'tropical_very_dry'
        elif 1000 <= precipitation < 2000:
            out = 'tropical_dry'
        elif 2000 <= precipitation < 4000:
            out = 'tropical_moist'
        elif 4000 <= precipitation < 8000:
            out = 'tropical_wet'
        elif 8000 <= precipitation:
            out = 'tropical_rain'
        else:
            pass
    elif 1000 <= altitude < 2000:
        if 500 <= precipitation < 1000:
            out = 'premontane_dry'
        elif 1000 <= precipitation < 2000:
            out = 'premontane_moist'
        elif 2000 <= precipitation < 4000:
            out = 'premontane_wet'
        elif 4000 <= precipitation < 8000:
            out = 'premontane_rain'
        else:
            pass
    elif 2000 <= altitude < 3000:
        if 500 <= precipitation < 1000:
            out = 'lower_montane_dry'
        elif 1000 <= precipitation < 2000:
            out = 'lower_montane_moist'
        elif 2000 <= precipitation < 4000:
            out = 'lower_montane_wet'
        elif 4000 <= precipitation < 8000:
            out = 'lower_montane_rain'
        else:
            pass
    elif 3000 <= altitude < 4000:
        if 500 <= precipitation < 1000:
            out = 'montane_moist'
        elif 1000 <= precipitation < 2000:
            out = 'montane_wet'
        elif 2000 <= precipitation < 4000:
            out = 'montane_wet'
        else:
            pass
    else:
        pass
```

```
49
50      return out
```

6. Lectura de los archivos raster

Gdal lee los archivos a través de la función `Open` (código 2, lineas 11–13). Cuando se ejecuta dicha función se retorna un objeto de archivo, que nos permitirá posteriormente acceder a cada uno de los componentes del mismo.

Existen varias formas de acceder a los metadatos de los archivos con gdal. Una de las más simples es a través de las matrices affine. El método `GetGeoTransform()` obtiene la matriz affine asociada a un objeto de archivo raster. Dicha matriz tiene 6 elementos, el origen geográfico está contenido en el primer y cuarto elementos, mientras que el tamaño del pixel en el segundo y sexto elementos (líneas 15–19).

Para leer los datos de cada archivo y mantenerlos en memoria se debe accesar cada banda con el método `GetRasterBand()` (líneas 27, 29 y 32) y posteriormente se convierten en arrays de Numpy con el método `ReadAsArray()` (líneas 33). La función `GetRasterBand()` requiere como único parámetro el número de la banda que se quiere leer. En el método `ReadAsArray()` solo se especificaron los cuatro primeros argumentos: el offset de lectura en el eje X, offset de lectura en el eje Y, y número de columnas y filas a leer. Estos dos últimos están contenidos en los atributos `RasterXSize` y `RasterYSize` del objeto del archivo.

Para facilitar el conteo de pixeles se utilizará un diccionario: los nombres de las zonas de vida serán empleados como llaves, y los conteos de los correspondientes pixeles serán empleados como valores (línea 9).

Código 2: Lectura de los archivos raster a traves de la librería `gdal`.

```
1  import gdal
2
3  alt_file = "vent_alt.tif"
4
5  prec_file = "precp.tif"
6
7  for_file = "BQNBQ_2016_EPSG4326.tif"
8
9  holdrigde_count = {}
10
11 alt_ras = gdal.Open(alt_file)
12 prec_ras = gdal.Open(prec_file)
13 for_ras = gdal.Open(for_file)
14
15 transform = alt_ras.GetGeoTransform()
16 altXOrigin = transform[0]
17 altYOrigin = transform[3]
18 altPixelWidth = transform[1]
19 altPixelHeight = transform[5]
20
```

```
21 transform = for_ras.GetGeoTransform()
22 forXOrigin = transform[0]
23 forYOrigin = transform[3]
24 forPixelWidth = transform[1]
25 forPixelHeight = transform[5]
26
27 for_band = for_ras.GetRasterBand(1)
28
29 alt_band = alt_ras.GetRasterBand(1)
30 alt_arr = alt_band.ReadAsArray(0,0,alt_ras.RasterXSize, alt_ras.
      RasterYSize)
31
32 prec_band = prec_ras.GetRasterBand(1)
33 prec_arr = prec_band.ReadAsArray(0,0,prec_ras.RasterXSize, prec_ras.
      RasterYSize)
```

7. Iteración por pixeles

La iteración de arrays de Numpy se puede maximizar utilizando los iteradores de Numpy, los cuales son más rápidos que los iteradores por defecto de Python. En este ejercicio utilizaremos el iterador `nditer`. El primer parámetro del iterador corresponde a los arrays que se van a iterar, mientras con la opción `flags` se pueden especificar varias opciones adicionales. En este caso usamos la opción "multiindex" que permite acceder a los índices de todos los niveles del array al interior de cada repetición (línea 3).

Una vez se inicializa el objeto del iterador, se ejecuta la iteración con el atributo `finished` (línea 4). El iterador se inicializa en la primera posición de los arrays, y para observar el siguiente par de elementos se utiliza el método `iternext()` (línea 28).

Frecuentemente los archivos raster contienen pixeles sin valores, los cuales son codificados usualmente como números negativos muy grandes. Dado que dichos valores no son de utilidad en nuestro cálculo se debe realizar una prueba condicional al respecto (línea 5).

Después de calcular la zona de vida (línea 6) se realiza un test. Algunas combinaciones de altitud y elevación no producen zonas de vida (p.e., altitudes mayores a 4000 m.), por lo cual es necesario comprobar si la función `holdridge` si retorna algún valor (línea 8). Si ésta prueba es positiva, se consultan las coordenadas geográficas del pixel empleando el índice del pixel en el array y los metadatos del archivo (líneas 9–12).

Una vez se adquieren las coordenadas es necesario consultar si el raster de bosque-no bosque contiene información para las mismas (línea 14). De ser así, se convierten las coordenadas en posiciones de la tabla de valores del archivo raster (líneas 15–16) y se confirma que efectivamente existan valores (línea 18). Finalmente, se accede a los valores de la banda de bosque-no bosque con el método `ReadAsArray` (línea 19) y se modifica el conteo en la estructura de datos que se declaró para tal fin (líneas 21–26).

Código 3: Iteración simultanea de arrays de Numpy y cálculos a nivel de pixel.

```
1 import numpy as np
```

```
2
3  it = np.nditer((alt_arr, prec_arr), flags=['multi_index'])
4  while not it.finished:
5     if it[0] >= 0 and it[1] > 0:
6        holdr = holdridge(it[0], it[1])
7
8        if holdr:
9           row = it.multi_index[0]
10          col = it.multi_index[1]
11          lon = col * altPixelWidth + altXOrigin
12          lat = row * altPixelHeight + altYOrigin
13
14          if lon >= forXOrigin and lat <= forYOrigin:
15             fpx = int((lon - forXOrigin) / forPixelWidth)
16             fpy = int((lat - forYOrigin) / forPixelHeight)
17
18             if fpx <= for_ras.RasterXSize and fpy <= for_ras.
                  RasterYSize:
19                for_val = for_band.ReadAsArray(fpx, fpy, 1, 1)
20
21                if for_val and for_val[0][0] == 1:
22
23                   if not holdr in holdrigde_count:
24                      holdrigde_count[holdr] = 1
25                   else:
26                      holdrigde_count[holdr] += 1
27
28    it.iternext()
```