geographica

Introducción al procesamiento de datos geográficos con Python

Alberto Asuero - @alasarr
Cayetano Benavent - @cayetanobv

MeetUp Python Sevilla - Enero 2016

Sumario de contenidos

Introducción

*“**Tobler's first law of geography***

Everything is related to everything else, but near things are more related than distant things.”

(Waldo Tobler, 1970)

Introducción

"What is geoprocessing?

Geoprocessing is a general term for manipulating spatial data, whether raster or vector. As you can imagine, that covers an awful lot of ground.

(...)

You even use geoprocessing in your daily life, whether you realize it or not.

(...)

Although some geoprocessing techniques are rather complicated, many are fairly simple. (...)"

(Garrard, 2016)

geographica

El Sistema de referencia de coordenadas: PROJ.4

PROJ.4 = Cartographic Projections Library

➔ Librería base para manejar Sistemas de Referencia de Coordenadas (CRS).

➔ Permite tanto conversiones entre distintas proyecciones cartográficas, como cálculos y transformaciones geodésicas.

➔ Una parte muy importante del geo-open source software (y mucho no open) utilizan Proj.4: QGIS, GRASS GIS, MapServer, PostGIS, Mapnik, GDAL, etc.

El Sistema de referencia de coordenadas: PROJ.4

- Esta librería ha sido liberada bajo la MIT license.
  La web oficial del proyecto: https://github.com/OSGeo/proj.4/wiki

- Está íntegramente escrita en el lenguaje C.

- Puede usarse a través de la API C o a través de su CLI.

El Sistema de referencia de coordenadas: PROJ.4

PROJ.4 - Python Binding

➔ Para manejar Proj.4 directamente en Python, existe la librería Pyproj, escrita por J. Whitaker (NOAA – ESRL).

➔ La librería tiene dos grandes partes:

PROJ: conversiones entre proyecciones cartográficas.

GEOD: para efectuar cálculos geodésicos.

geographica

El Sistema de referencia de coordenadas: PROJ.4

PROJ.4 - Python Binding

➔ Ejemplo de uso de la librería Pyproj (uso de la clase Proj para reproyectar coordenadas):

http://nbviewer.jupyter.org/github/GeographicaGS/MeetUp_Python_Jan2016/blob/master/code/notebooks/pyproj_examples.ipynb

El Sistema de referencia de coordenadas: PROJ.4

- Es muy importante saber la versión que estamos usando. La algoritmia del módulo de Geodesia cambia radicalmente entre las versiones 4.8 y 4.9.

- La versión 4.9 utiliza un port a C del módulo Geodesic de la librería GeographicLib de C. Karney. Es una librería muy sólida y precisa.

- Breve discusión y comparativa (desde Python) entre versiones de Proj4 y otras librerías, en un pequeño benchmark: https://github.com/cayetanobv/GeodeticMusings

El Sistema de referencia de coordenadas: PROJ.4

- Ejemplos de uso de la librería Proj.4 desde Python (junto a otras librerías que veremos más adelante):

  https://github.com/GeographicaGS/GeodesicLinesToGIS

Fig 1. Mercator Projection.

Fig 4. Geodesic lines crossing 180°, Mercator Projection (Centered on 180°E).

Leer y escribir datos espaciales: GDAL

GDAL - Geospatial Data Abstraction Library

- Leer y escribir datos GIS espaciales, en el mundo del software libre, es casi sinónimo de trabajar con GDAL: http://www.gdal.org/

- Desde la v1.3.2, GDAL y OGR se unen en un mismo proyecto (pertenecen al mismo “source tree”), y pasa a llamarse GDAL.

- GDAL es la capa de lectura/escritura de casi todo el software libre geo (¡y de mucho software no libre!):
  https://trac.osgeo.org/gdal/wiki/SoftwareUsingGdal

Leer y escribir datos espaciales: GDAL

- GDAL se distribuye bajo una licencia de estilo MIT/X: https://trac.osgeo.org/gdal/wiki/FAQGeneral#WhatlicensedoesGDALOGRuse

- Está escrita en C/C++, y su API principal, obviamente, se accede desde C/C++.

- Hay APIs en otros lenguajes. Las activas actualmente son: C#, Java, Perl, and Python. Otras APIs a GDAL externas al proyecto (“outside of the GDAL source tree”): Go, Lua, Node.js, R, Tcl, etc.

Leer y escribir datos espaciales: GDAL

- GDAL soporta actualmente (enero 2016):

  142 formatos raster → http://www.gdal.org/formats_list.html

  84 formatos vectoriales → http://www.gdal.org/ogr_formats.html

- GDAL permite escritura en una gran parte de los formatos soportados (no todos). Muchos formatos requieren compilación específica de ciertos drivers (o librerías dependientes).

- Con cada versión nueva, se añaden multitud de formatos y nuevas capacidades (¡revisar bien CHANGELOG!).

Leer y escribir datos espaciales: GDAL

- Con la versión actualmente estable (GDAL 2.0), la API cambia por completo. Su integración con OGR es ya muy elevada.

- Hay que tener en cuenta que la mayor parte del software libre que depende de GDAL para leer o escribir (casi todo!!), aún está en proceso de adptación al cambio de v1.X a v2.0.

- Para ver un resumen con las principales novedades, recomiendo leer la presentación de Even Roualt (actual PSC Chair) del FOSS4Geo2015:

  http://download.osgeo.org/gdal/presentations/GDAL%202.0%20overview%20(FOSS4G-E%202015).pdf

Leer y escribir datos espaciales: GDAL

➔ Antes de empezar a trabajar con GDAL, siempre debemos asegurarnos de dos cosas:

1. Versión de GDAL instalada:

```
$ gdalinfo --version
```

2. Formatos soportados en la versión que tenemos instalada:

```
$ gdalinfo --formats
```

2 (bis). Ver detalles de formatos concretos:

```
$ gdalinfo --format GTIFF
```

Leer y escribir datos espaciales: GDAL

GDAL - Python API

- La API de Python es una de las más utilizadas. La wiki oficial es:
  https://trac.osgeo.org/gdal/wiki/GdalOgrInPython

- GDAL suele ser la primera capa de cualquier desarrollo o flujo de geoprocesamiento que diseñemos para trabajar con datos GIS.

- La documentación de la API (generada automáticamente):
  http://gdal.org/python/

Leer y escribir datos espaciales: GDAL

- GDAL provee un conjunto de utilidades de línea de comando muy potentes:

  http://www.gdal.org/gdal_utilities.html

- Estas utilidades, escritas en C/C++ o en Python, son excelentes ejemplos de cómo manejar la API de GDAL.

- Los test de GDAL están casi todos escritos en Python, por lo que estudiarlos es otra manera de aprender: https://svn.osgeo.org/gdal/trunk/autotest/

Leer y escribir datos espaciales: GDAL

➔ importante tener en cuenta que hay numerosos “Gotchas” si usamos GDAL desde Python.

➔ Ejemplos (algunos sorprendentes para programadores expertos en Python):

- Python bindings do not raise exceptions unless you explicitly call UseExceptions().

- Certain objects contain a Destroy() method, but you should never use it.

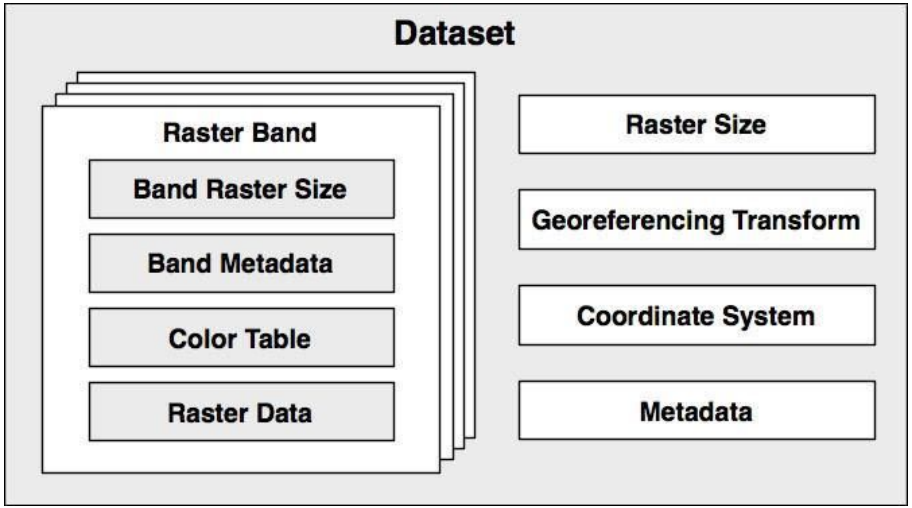
- etc.

https://trac.osgeo.org/gdal/wiki/PythonGotchas

Leer y escribir datos espaciales: GDAL

➔ Modelo de datos raster en GDAL:

Detalles en http://www.gdal.org/gdal_datamodel.html



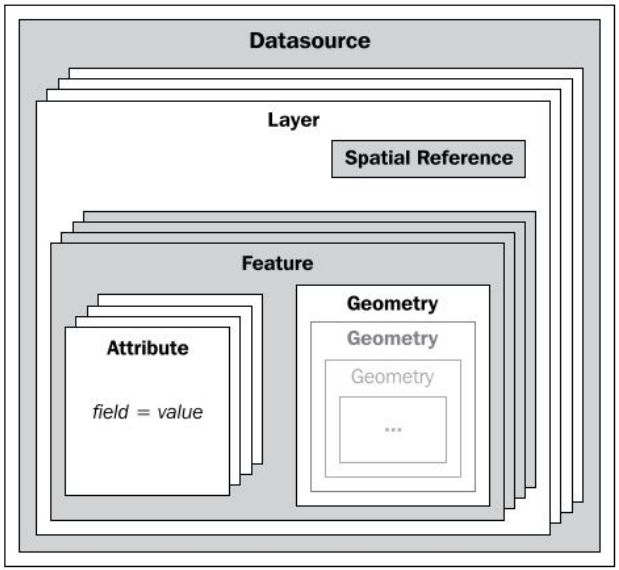
(E. Westra, 2014)

geographica G

Leer y escribir datos espaciales: GDAL

➔ Modelo de datos vectorial en GDAL (OGR):

Detalles en http://www.gdal.org/ogr_arch.html



(E. Westra, 2014)

geographica

Leer y escribir datos espaciales: Rasterio y Fiona

- ➔ Rasterio y Fiona son una excelente alternativa para leer y escribir datos raster y vectoriales desde Python.
- ➔ No son un wrapper al binding de Python. Construidos sobre GDAL con Cython.
- ➔ Es más sencilla de manejar que el binding “oficial”, más rápida y mucho más Pythónica.

Leer y escribir datos espaciales: Rasterio y Fiona

➔ Lecturas interesantes para aprender Rasterio y Fiona:

http://sgillies.github.io/foss4g-2014-fiona-rasterio

https://github.com/sgillies/frs-cheat-sheet

“- A Python package at the top.

- Extension modules (using Cython) in the middle.

- Fast loops, typed memoryviews, "nogil" blocks.

- GDAL shared library on the bottom.”

Análisis espacial raster

Álgebra de mapas

➔ Es el lenguaje de análisis ráster por excelencia.

➔ Permite realizar análisis y modelado de la información geográfica a través de expresiones algebraicas (no hay que olvidar que una capa ráster está formada por celdas, cada una de las cuales tiene un valor numérico con el que podemos operar algebraicamente).

Análisis espacial raster: GDAL y Rasterio

GDAL y/o Rasterio + NumPy

➔ Una vez hemos abierto un fichero raster como Dataset en GDAL y/o Rasterio, lo que hace tremendamente potente a Python, es poder leerlo como Numpy Array.

Análisis espacial raster: GDAL y Rasterio

GDAL y/o Rasterio + NumPy

- ➔ Todo el poder de Numpy en nuestras manos para hacer cómputos vectorizados muy rápidos con el Dataset raster.

- ➔ Obviamente, ello hace que el Dataset esté expuesto a todo el Stack científico de Python (Scipy, Matplotlib, Scikits, etc.).

- ➔ Tras operar con Numpy con el Dataset, podemos escribir a un formato raster con GDAL el resultado.

Análisis espacial raster: GDAL y Rasterio

➔ Para aprender más sobre GDAL y Rasterio, puede accederse al workshop:

https://github.com/GeographicaGS/workshop_Raster_GIS_data

Análisis espacial vectorial: GEOS

GEOS - Geometry Engine, Open Source

GEOS Geometry Engine Open Source

- GEOS permite realizar análisis espacial complejo sobre datos vectoriales.
- GEOS es uno de los proyectos principales de OSGeo, y está liberado bajo la licencia GNU LGPL.
- La página del proyecto: https://trac.osgeo.org/geos/

Análisis espacial vectorial: GEOS

GEOS - Geometry Engine, Open Source

GEOS Geometry Engine Open Source

- Es un port a C++ de la librería JTS (Java Topology Suite).

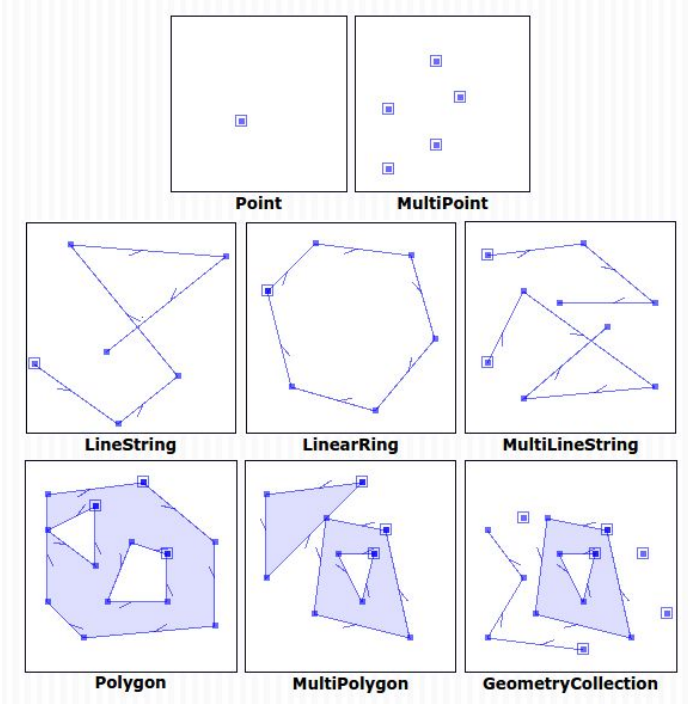
- Implementa los predicados espaciales del Simple Features Specification for SQL definido por la OGC, además de otras funciones topológicas avanzadas.

Análisis espacial vectorial: GEOS

GEOS Geometry Engine Open Source

➔ Spatial Data Model

Geometries in GEOS have an Interior, a Boundary, and an Exterior.



Fuente: Vividsolutions.

geographica G

Análisis espacial vectorial: GEOS

GEOS Geometry Engine Open Source

- Capacidades de GEOS:

  - Geometries: Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, GeometryCollection

  - Predicates: Intersects, Touches, Disjoint, Crosses, Within, Contains, Overlaps, Equals, Covers

  - Operations: Union, Distance, Intersection, Symmetric Difference, Convex Hull, Envelope, Buffer, Simplify, Polygon Assembly, Valid, Area, Length,

  - Prepared geometries (pre-spatially indexed)

  - STR spatial index

  - OGC Well Known Text (WKT) and Well Known Binary (WKB) encoders and decoders.

  - C and C++ API (C API gives long term ABI stability)

  - Thread safe (using the reentrant API)

geographica G

Análisis espacial vectorial: GEOS

GEOS Geometry Engine Open Source

- GEOS está actualmente (enero 2016) en su versión 3.5.
- Es multiplataforma, y tienes dos API oficiales: C (recomendada) y C++.
- GEOS es un pilar clave del geo open source, formando parte de muchas aplicaciones:

  QGIS, GDAL, PostGIS, GRASS GIS, MapServer, SAGA GIS, etc.

Análisis espacial vectorial: Shapely

¿Python Binding?

➔ No hay actualmente (antes sí lo había) un binding de Python dentro de GEOS. Desde el proyecto indican cómo usar GEOS desde Python:

> *Since version 3.0 the Python bindings are no longer supported. Options:*
>
> *1) Become or recruit a new maintainer.*
>
> *2) Use **Shapely** (http://pypi.python.org/pypi/Shapely) on Python 2.4+*
>
> *3) Simply call functions from libgeos_c via Python ctypes.*

https://trac.osgeo.org/geos/ticket/228

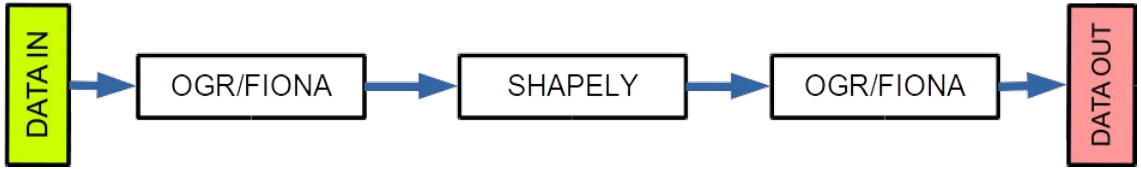
Análisis espacial vectorial: Shapely

- La opción preferente para usar GEOS en Python es la librería Shapely.
- Es una interfaz madura, estable, muy optimizada (con Cython) y muy Pythónica.
- Nos permite hacer operaciones “PostGIS-like” sin tener que tener una RDBMS y lo más importante, combinar GEOS con el stack científico de Python (Numpy, Scipy, etc.).

Análisis espacial vectorial: Shapely

- Shapely no maneja (ni se preocupa) de formatos de datos; únicamente trabaja con objetos geométricos.

- En los flujos de geoprocesamiento, necesitamos que los data input/output (a distintos formatos) los gestione GDAL (OGR) y/o Fiona.



Análisis espacial vectorial: Shapely

Shapely + Numpy

➔ Todos los objetos geométricos (con secuencias de coordenadas) tienen una interfaz con Numpy.

➔ Podemos ver un ejemplo en:

http://nbviewer.jupyter.org/github/GeographicaGS/MeetUp_Python_Jan2016/blob/master/code/notebooks/shapely_numpy_examples.ipynb

Ejemplo de renderización cartográfica: Mapnik

mapnik

- A continuación, veremos un ejemplo de renderización cartográfica con Mapnik:

  http://mapnik.org/

- Mapnik es un motor de renderizado cartográfico open source, escrito en C++ (con binding a Python).

geographica G

Ejemplo de renderización cartográfica: Mapnik

mapnik

- Este ejemplo se llevará a cabo con la librería Equidna:

  https://github.com/GeographicaGS/Equidna

- Equidna es un motor de tileado de mapas escrito en Python sobre Mapnik. Permite una fuerte compresión de la salida y puede paralelizarse el cómputo de tiles.

- Su output es un fichero MBTiles.

geographica G

Ejemplo de renderización cartográfica: Mapnik

mapnik

- El dataset ejemplo a renderizar será generado con la librería declinationmap:

  https://github.com/cayetanobv/declinationmap

  “Declination map

  Computing declination data from World Magnetic Model for entire world or for a given bounding box. Outputs are GIS files: raster (GeoTiff) and vector (Shapefile).”

geographica G

Tiles

geographica

Tileando Mapas

- Images tiles
- Vector tiles

Ejemplos - CartoDB

http://demo-routes.geographica.gs/

Documentación

➔ Toda la documentación en:

https://github.com/GeographicaGS/MeetUp_Python_Jan2016

¡¡Gracias y hasta la próxima!!

geographica