Работа с растрами при помощи GDAL и Python

Обсудить в форуме Комментариев — 15

Эта страница опубликована в основном списке статей сайта
по адресу http://gis-lab.info/qa/gdal-python.html

Руководство по использованию GDAL Python API, частично основано на «GDAL API tutorial»

GDAL — свободная библиотека для работы с растровыми данными. Утилиты командной строки, входящие в состав библиотеки широко используются для выполнения разнообразных задач.

Благодаря наличию развитого API можно работать с функциями GDAL из многих языков программирования. та статья описывает работу с GDAL API через Python. При написании статьи использовались материалы GDAL API tutorial.

Содержание

Подготовка

Будем исходить из того, что все программное обеспечение установлено при помощи установщика OSGeo4W. Для использования GDAL совместно с Python необходимо наличие соответствующий байндингов (bindings). В стеке программ OSGeo4W нужный пакет называется gdal-python (для GDAL 1.5.x) или gdal16-python (для GDAL 1.6.x). Рекомендую использовать версию 1.6.x. Кроме того, для выполнения различных операций с растром, нам потребуется пакет numpy.

GDAL Python API состоит из пяти основных модулей и пяти дополнительных (существуют для совместимости со старыми версиями):

- gdal — Python интерфейс к библиотеке GDAL
- ogr — Python интерфейс к библиотеке OGR
- osr — работа с системами координат
- gdal_array — вспомогательные функции
- gdalconst — константы

Подключить их можно командами

```
# основные
from osgeo import gdal
from osgeo import ogr
from osgeo import osr
from osgeo import gdal_array
from osgeo import gdalconst

# версии для совместимости. Будут удалены в версии 2.0
import gdal
import ogr
import osr
```

```
import gdalnumeric
import gdalconst
```

Если используется GDAL версии 1.5 и выше, рекомендуется использовать «основные» модули. А для случаев, когда необходимо использовать код написанный ранее можно сделать проверку

```
try:
  from osgeo import gdal
except ImportError:
  import gdal
```

В большинстве случаев достаточно подключить только модуль gdal.

Открытие файла

Для открытия растра используем функцию gdal.Open(), в качестве аргумента принимающую полный путь к растру и необзятельную константу, описывающую режим открытия. Если константа опущена, то подразумевается режим только для чтения. В случае успеха функция вернет объект GDALDataset, в противном случае — None.

```
import osgeo.gdal as gdal
gdalData = gdal.Open( "/home/alex/test/input.tiff", GA_ReadOnly )
# или так
# gdalData = gdal.Open( "/home/alex/test/input.tiff" )

# проверяем все ли в порядке
if gdalData is None:
  print "ERROR: can't open raster"
  sys.exit( 1 )
```

Получение информации

Объект GDALDataset позволяет получить информацию о количестве каналов растра, извлечь данные и метаданные. Все используемые функции имеют понятные названия, поэтому подробно их описывать не вижу смысла, вместо этого приведу небольшой пример:

```
print "Driver short name", gdalData.GetDriver().ShortName
print "Driver long name", gdalData.GetDriver().LongName
print "Raster size", gdalData.RasterXSize, "x", gdalData.RasterYSize
print "Number of bands", gdalData.RasterCount
print "Projection", gdalData.GetProjection()
print "Geo transform", gdalData.GetGeoTransform()
```

Последняя функция возвращает очередь из 6 чисел:

- X координата левого верхнего пикселя
- ширина пикселя
- вращение
- Y координата левого верхнего пикселя
- вращение
- высота пикселя

Извлечение растра/канала

Любой растр можно представить в виде массива, одноканальный в виде двумерного размерностью X на Y; многоканальный — в виде многомерного. GDAL позволяет получить доступ как ко всему растру сразу, так и к любому каналу. Результатом такой операции будет массив numpy соответсвующей размерности.

```
# получаем весь растр целиком
raster = gdalData.ReadAsArray()
# получаем отдельный канал
gdalBand = gdalData.GetRasterBand( 1 )
band_1 = gdalBand.ReadAsArray()
```

Кроме того, можно обрабатывать растр построчно или блоками

```
line = gdalBand.ReadAsArray( xoffeset, yoffset, xsize, ysize )
```

Здесь

- xoffset — смещение по X
- yoffset — смещение по Y
- xsize — размер блока по X
- ysize — размер блока по Y

Т.е. если оба смещения буду равны нулю, а xsize и ysize — высоте и ширине растра, то в переменной line мы получим первую строку массива. А изменяя значение yoffset можно последовательно пройтись по всем строкам массива.

При извлечении данных следует помнить, что тип данных в массиве соответсвует типу данных растра, и в некоторых случаях это может привести к ошибкам. Например, при вычитании двух целочисленных беззнаковых массивов вполне можно получить неправильный результат из-за целочисленного переполнения. Поэтому при извлечении данных нужно быть внимательными и при необходимости выполнять приведение к другому типу данных. Например, следующая строка извлечет канал растра и принудетельно установит тип данных в Float32 (32 разрядное число с плавающей точкой)

```
band_1 = gdalBand.ReadAsArray().astype( numpy.float32 )
```

Аналогично выполняется приведение к любому другому типу данных. Полный список поддерживаемых типов данных можно найти в документации к numpy.

Операции с растром

Для работы с массивами (а растр это тот же массив) удобно использовать пакет numpy — расширение, добавляющее поддержку больших многомерных массивов и матриц, а также обширную библиотеку математических функций для работы с этими массивами.

Практически все функции в numpy работают поэлементно, что значительно упрощает процесс обработки массивов. Так, для сложения двух массиво одной размерности достаточно следующей строки

```
result = numpy.add( array1, array2 )
```

Аналогично выполняется вычитание, умножение, деление и другие действия. Унарные операции тоже поэлементные, например после выполнения сроки

```
r = sin( array1 )
```

в массив r будут записаны значения синуса каждого элемента массива array1. Вот более сложный пример — поэлементная (попиксельная) обработка растра

```
gdalData = gdal.Open( "/home/alex/test/input.tiff" )
# размер растра
xsize = gdalData.RasterXSize
ysize = gdalData.RasterYSize
# получаем растр в виде массива
raster = gdalData.ReadAsArray()
# перебираем все пиксели растра
for col in range( xsize ):
  for row in range( ysize ):
    # если значение пикселя равно 5, то меняем его на 10
    # иначе значение остается без изменений
    if raster[ row, col ] == 5:
      raster[ row, col ] = 10
```

Тот же цикл поиска-замены значения, но с использованием значительно более быстрых функций обработки

массивов Numpy выглядит следующим образом:

```
temp1_bool = numpy.equal(raster,5)
numpy.putmask(raster,temp1_bool,10)
```

Сохранение файла

Существует два способа сохранить растр: используя CreateCopy() или Create(). При использовании CreateCopy() необходимо указать растр-источник, параметры которого будут использованы при создании нового растра. В случае использования Create() необходимо вручную сформировать метаданные и выполнить запись растра.

Следует помнить, что не все драйверы поддерживают метод Create(), поэтому вначале необходимо проверить, есть ли поддержка нужного метода в драйвере:

```
format = "GTiff"
driver = gdal.GetDriverByName( format )
metadata = driver.GetMetadata()
if metadata.has_key( gdal.DCAP_CREATE ) and metadata[ gdal.DCAP_CREATE ] == "YES":
  pass
else:
  print "Driver %s does not support Create() method." % format
  sys.exit( 1 )
# аналогично выполняется проверка для CreateCopy
if metadata.has_key( gdal.DCAP_CREATECOPY ) and metadata[ gdal.DCAP_CREATECOPY ] ==
"YES":
  pass
else:
  print "Driver %s does not support CreateCopy() method." % format
  sys.exit( 1 )
```

Некоторые драйверы могут работать в режиме только чтения и не поддерживают ни один из этих методов.

При использовании метода CreateCopy() вся необходимая информация берется из «эталонного» растра, но есть возможность задать специфичные для формата параметры

```
# эталонный растр
inputData = gdal.Open( "/home/alex/test/input.tiff" )
# создаем свой растр "по образу и подобию" inputData
outputData = driver.CreateCopy( "/home/alex/test/output.tiff", inputData, 0 )
# то же самое, но заданы дополительные параметры
# outputData = driver.CreateCopy( "/home/alex/test/output.tiff", inputData, 0,
'TILED=YES', 'COMPRESS=PACKBITS'  )
# закрываем датасеты и освобождаем память
inputData = None
outputData = None
```

При использование Create() необходимо вручную задать количество каналов, размеры растра и желаемый тип данных (байт, длинное целое, число с плавающей запятой...)

```
# размеры растра
cols = 512
rows = 512
# количество каналов
bands = 1
# тип данных
dt = gdal.GDT_Byte
# создаем растр
outData = driver.Create( "/home/alex/test/out.tiff", cols, rows, bands, dt )
```

После того, как растр создан можно добавить информацию о проекции.

```
outData.SetProjection( proj )
outData.SetGeoTransform( transform )
```

И записать данные в канал

```
outData.GetRasterBand( 1 ).WriteArray( raster )
# после записи данных закрываем датасет
outData = None
```

Если каналов несколько, записывать удобнее в цикле

```
bands = 3
for i in range( bands ):
  outData.GetRasterBand( i + 1 ).WriteArray( raster[ i ] )
outData = None
```

Само собой, массив raster должен иметь столько измерений, сколько каналов в растре (или больше).

Пример. Простой растровый калькулятор

Вооружившись этими сведениями попробуем написать простенький скрипт на Python для сложения двух каналов растра. Скрипт принимает два параметра: исходный растр и путь для сохранения результата.

```
# -*- coding: utf-8 -*-

#!/usr/bin/env python

import sys
import numpy
import osgeo.gdal as gdal

# функция проверяющая растр на многоканальность
def isMultiband( path ):
  gdalData = gdal.Open( path )
  if gdalData.RasterCount < 2:
    print "ERROR: raster must contain at least 2 bands"
    return False
  return True

# извлекает из заданного растра канал с заданным номером
def bandAsArray( path, bandNum ):
  gdalData = gdal.Open( path )
  gdalBand = gdalData.GetRasterBand( bandNum )
  array = gdalBand.ReadAsArray().astype( numpy.float32 )
  gdalBand = None
  gdalData = None
  return array

# сохраняет массив array как растр с именем outPath в формате
# GeoTiff. Данные о проекции берутся из растра etalonPath
def saveRaster( outPath, etalonPath, array ):
  gdalData = gdal.Open( etalonPath )
  projection = gdalData.GetProjection()
  transform = gdalData.GetGeoTransform()
  xsize = gdalData.RasterXSize
  ysize = gdalData.RasterYSize
  gdalData = None

  format = "GTiff"
  driver = gdal.GetDriverByName( format )
  metadata = driver.GetMetadata()
  if metadata.has_key( gdal.DCAP_CREATE ) and metadata[ gdal.DCAP_CREATE ] == "YES":
    outRaster = driver.Create( outPath, xsize, ysize, 1, gdal.GDT_Float32 )
    outRaster.SetProjection( projection )
    outRaster.SetGeoTransform( transform )
    outRaster.GetRasterBand( 1 ).WriteArray( array )
    outRaster = None
```

```
  else:
    print "Driver %s does not support Create() method." % format
    return False

if __name__ == '__main__':
  args = sys.argv[ 1: ]
  inPath = args[ 0 ]
  outPath = args[ 1 ]
  if isMultiband( inPath ):
    band1 = bandAsArray( inPath, 1 )
    band2 = bandAsArray( inPath, 2 )
    res = numpy.add( band1, band2 )
    if not saveRaster( outPath, inPath, res ):
      print "ERROR: saving failed"
      sys.exit( 1 )
```

Ссылки по теме

- Работа с векторными данными при помощи OGR и Python
- GDAL API tutorial
- GDAL C API
- GDAL/OGR in Python
- Tentative NumPy Tutorial
- NumPy and SciPy documentation

Обсудить в форуме Комментариев — 15

Последнее обновление: 2014-05-15 00:14

Дата создания: 01.04.2010
Автор(ы): Александр Бруй