Создание скрипта на Python для ГИС GRASS

Обсудить в форуме Комментариев — 9

Эта страница опубликована в основном списке статей сайта
по адресу http://gis-lab.info/qa/grass64-python.html

Как сделать новый модуль для GRASS

В статье рассматривается пример того, как можно расширить функциональность ГИС GRASS под нужды пользователя. В качестве примера возьмем задачу создания скрипта, вычисляющего расстояние Джеффриса-Мацуситы - меры, оценивающей степень различия классов. В качестве языка реализации был выбран язык Python, который в настоящее время становится одним из основных языков обработки геоданных.

Цель данной статьи состоит не в том, чтобы объяснить, как работает расстояние реализуемое этим конкретным скриптом, а в том, чтобы показать основные моменты, возникающие при создании собственного расширения: как производится передача параметров скрипту, как выполнить вызов внешних модулей и как получить результаты их работы.

В данной статье рассматривается подход, применимый к ГИС GRASS текущей стабильной версии 6.4. Начиная с версии GRASS GIS 7.0, которая в данный момент находится в стадии тестирования, все используемые системой скрипты, будут переведены на язык Python, соответственно, описанный ниже подход останется в силе и для последующих версий. Однако, следует оговориться, что версия GRASS GIS 7.0 предоставляет гораздо более широкие возможности для программирования расширений на языке Python, чем текущая версия, но эти возможности в данной статье не рассматриваются.

Содержание

Общий обзор

Вызов скрипта

Сначала кратко пробежимся по основным моментам, а во втором разделе расмотрим реализацию более детально.

Начнём с того, что определимся с входными и выходными данными. Как представляется, от нашего скрипта логично потребовать того, что он будет принимать на входе:

- список изображений (растров), из которых будут извлекаться значения яркостей пикселей для последующей классификации;

- растр, в котором хранятся маски объектов различных классов;
- два числа - номера тех категорий (классов), для которых нужно рассчитать расстояние.

Также логично потребовать от скрипта, чтобы он выдавал на выходе число - значение расстояния Джефриса-Мацуситы для заданных классов.

Таким образом, запуск скрипта будет выглядеть так:

```
GRASS> i.jmdist base=A2000.092.1,A2000.092.2,A2000.104.1,A2000.104.2 cover=class
cats=1,2
```

где i.jmdist - название скрипта, base - параметр, в котором передается список анализируемых растров, cover - название растра масок, cats - список категорий.

Формулы расчетов

В качестве справки в данном подразделе приводятся формулы, по которым производятся вычисления.

Расстояние будет рассчитываться по формуле:

$$J_{ij} = \sqrt{2(1 - e^{-B})}$$

где B -- расстояние Бхаттачариа

$$B = \frac{1}{8}MH + \frac{1}{2}\ln\left\{\frac{|(\sum_i + \sum_j)/2|}{|\sum_i|^{1/2}|\sum_j|^{1/2}}\right\},$$

которое, в свою очередь, расчитывается на основе расстояния Махалланобиса:

$$MH = \sqrt{(m_i - m_j)^t\left(\frac{\sum_i + \sum_j}{2}\right)^{-1}(m_i - m_j)},$$

где $\sum i$ и $\sum j$ --- ковариационные матрицы классов i и j соответственно, а m_i и m_j --- вектора средних данных классов.

Основные моменты реализации

Для того, чтобы рассчитать расстояние нам понадобится:

- вычислить векторы средних для каждого класса m_t;
- вычислить матрицы ковариаций для каждого класса $\sum t$;
- вычислить искомое расстояние.

Для того, чтобы найти вектор средних значений, можно воспользоваться модулем r.univar, который вычисляет различные статистики по растру. Для расчета матриц ковариаций существует модуль r.covar.

Следовательно, нужно написать модуль, который будет играть роль "клея", т.е. модуль, который будет:

- Считывать параметры base, cover, cats, передаваемые из командной строки.
- Строить маски и расчитывать статистики. (Сделать прозрачными все данные, которые не относятск к первой категории, рассчитать вектор средних и ковариации для первого класса; сделать прозрачными все данные, которые не относятся ко второй категории, расчитать вектор средний и ковариации для второго класса). Т.е.:
    - Построить соответствующую маску.
    - Вызвать модуль r.univar, передать ему название анализируемого(мых) растра(ов) и считать ответ (средние значения) в вектор средних.

    - Вызвать модуль r.covar и считать его ответ в матрицу ковариаций.
- После получения двух векторов стредних и двух матриц ковариаций, прозводить необходимые расчеты для вычисления искомого расстояния.

Здесь есть тонкое место: построение масок. Если не маскировать значения категорий, то расчеты будут производиться для всего растра в целом, а нам нужно получить соответствующие параметры для каждой категории в отдельности.

Для построения такого склеивающего модуля годится любой язык программирования, позволяющий запускать внешние программы. Здесь будет использован язык Python, соответственно, нужны знания о том, как Python производит создание и запуск подпроцессов.

Реализация

Для того, чтобы пользователям было удобно использовать язык Python для создания модулей для ГИС GRASS, разработчики этой системы предоставили пакет-обертку для основных функций GRASS. Поэтому, модуль, написанный под GRASS GIS на языке Python, должен сначала импортировать этот пакет (и, конечно же, другие пакеты, используемые модулем):

```
import os, sys                   # эти модули и функции
from math import exp, sqrt, log # пригодятся по ходу
import numpy as np               # вычислений.

import grass.script as grass   # это импорт возможностей GRASS GIS
```

Определение параметров, передаваемых модулю

Для того, чтобы указать, какие параметры и флаги будет использовать модуль, каковы их типы и т.д. вначале модуля необходимо прописать соответствующий заголовок:

```
#%Module
#% description: Program calculates Jeffries-Matusita Distance
#%End
#%option
#% key: base
#% type: string
#% gisprompt: base input maps
#% description: Name of base raster maps
#% required : yes
#% multiple: yes
#%end
#%option
#% key: cover
#% type: string
#% gisprompt: cover map
#% description: Name of cover raster map
#% required : yes
#% multiple: no
#%end
#%option
#% key: cats
#% type: integer
#% gisprompt: cats of cover map
#% description: Categories of cover raster map for analize
#% required : yes
#% multiple: yes
#%end
```

Что это означает? Рассмотрим на примере параметра base:

```
#%option
#% key: base
#% type: string
#% gisprompt: base input maps
#% description: Name of base raster maps
#% required : yes
#% multiple: yes
#%end
```

Здесь указаны основные свойства параметра, такие как название параметра (key), его тип (type), обязательный ли это параметр (required), может ли данный параметр содержать список значений (multiple).

В результате создаваемый модуль "будет знать", о том, что ему требуется для работы и даже сообщить об этом пользователю:

```
GRASS> i.jmdist --help

Description:
 Program calculates Jeffries-Matusita Distance

Usage:
 i.jmdist base=string,string,... cover=string cats=value,value,...
   --verbose --quiet

Flags:
 --v   Verbose module output
 --q   Quiet module output

Parameters:
   base   Name of base raster maps
  cover   Name of cover raster map
   cats   Categories of cover raster map for analize
```

Более того, если не будет указан какой-либо из обязательных параметров, то будет показано соответствующее сообщение:

```
GRASS> i.jmdist cover=class.435 cat=1,6

ERROR: Required parameter base not set:
    (Name of base raster maps).
```

Для того, чтобы получить доступ к передаваемым параметрам, необходимо использовать функцию parser():

```
if __name__ == "__main__":
        if "GISBASE" not in os.environ:
                print "You must be in GRASS GIS to run this program."
                sys.exit(1)

        options, flags = grass.parser()
        main(options, flags)
```

Эта функция анализирует параметры и флаги, передаваемые модулю, и возвращает словарь с соответствующими значениями, который может быть использован в дальнейшем:

```
def main(options, flags):
        ...
        base_maps = options'base'.split(',')
        cover_map = options'cover'
        cats = options'cats'.split(',')
        ...
```

Вызов модулей GRASS GIS из программы на Python

Для вызова существующих модулей GRASS GIS предоставляется функция run_command(), которой передаются параметры - название вызываемого модуля, а также флаги и параметры, используемые модулем. Например, если мы хотим воспользоваться модулем r.mask, управляющим слоем маски MASK, и установить маску, сделав прозрачными для системы все области, для которых значения растра class не равны 2:

```
GRASS> r.mask -o input=class maskcats=2
```

Для того, чтобы вызвать эту команду из Python, нужно передать соответствующие параметры функции run_command():

```
grass.run_command('r.mask', input='class', maskcats=2, flags='o')
```

Но часто возникает необходимость не только запустить определенный модуль с требуемыми параметрами, но и прочитать вывод этого модуля для дальнейшей обработки, так, например, нам потребуется результат расчетов матрицы ковариаций. Для этих целей можно воспользоваться идеей **каналов**, которые перенаправляют стандартный вывод одной программы на стандартный вход другой. Идея каналов является классической для UNIX-систем, но с недавних пор такая возможность появилась и в Windows. Чтение данных другой программы при помощи каналов предоставляет функция pipe_command().

До того, как мы рассмотрим вызов модуля r.covar из программы на Python, остановимся на том, какие параметры этот модуль требует для своей работы и каков его формат вывода.

Итак, модуль принимает параметр map - список названий растров, разделенных запятой, для которых рассчитываются ковариации, на выходе модуля - значение матрицы ковариаций:

```
GRASS> r.covar map=A2000.092.1,A2000.092.2,A2000.104.1,A2000.104.2

191239.578042 336787.737497 116070.882664 161302.532934
336787.737497 718625.461443 195030.439805 386349.959486
116070.882664 195030.439805 132796.215126 182562.501485
161302.532934 386349.959486 182562.501485 414655.863421
```

Таким образом, следующая функция вызывает модуль r.covar, считывает результаты его работы и возвращает список элементов матрицы ковариаций:

```
def get_covars(maps):
        """Вычисляет матрицу ковариаций для заданных карт"""
        result =
        p = grass.pipe_command('r.covar',map=maps, flags='q') # запустили модуль
r.covar
        for line in p.stdout:  # построчно считываем его стандартный вывод и разбиваем
строки, чтобы получить элементы матрицы
                result.append( float(a) for a in line.strip().split())
        p.wait()
```

```
        return result
```

Аналогично, для получения вектора средних значений для списка карт будет использоваться следующая функция:

```
def get_averages(maps):
        """Вычисляет средние значения ячеек для каждой карты из списка карт maps"""
        results =
        for m in maps:
                univar = {}
                p = grass.pipe_command('r.univar',flags='g',map=m)
                for line in p.stdout:
                        val,count = line.strip().split('=')
                        univarval = float(count)
                p.wait()
                results.append(univar'mean')
        return results
```

Полученные значения можно анализировать средствами Python, передавать их следующим модулям и т.д.

Результаты

Итак, в статье были рассмотрены следующие вопросы создания модуля для GRASS GIS на языке Python:

- Вызов модуля, анализ параметров командной строки.
- Запуск внешних программ при помощи функции run_command()
- Чтение результатов работы внешних программ при помощи функции pipe_command()

Поскольку цель статьи - дать читателю представление об основных моментах создания скриптов для GRASS GIS, то основной упор в статье делается на описание передачи данных между различными модулями GRASS, соответственно, в статье не разбирается функция вычисления расстояния. Заинтересовавшийся читатель может скачать созданный модуль по ссылке для более детального знакомства.

Ссылки по теме

- GRASS and Python

Обсудить в форуме Комментариев — 9

Последнее обновление: 2014-05-15 01:37

Дата создания: 25.04.2011
Автор(ы): Дмитрий Колесов, Максим Дубинин