Обработка данных на языке Python в ArcGIS

Обсудить в форуме Комментариев — 0

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

Пошаговая разработка скрипта для обработки данных использующего геопроцессинг ArcGIS

Разберем решение задачи расчета ошибок пользователя и производителя с помощью геопроцессинга ArcGIS и языка Python.

Эта статья демонстрирует необходимые шаги для решения задачи и может служить как учебный пример использования этой технологии. Для того, чтобы разобраться с тем, зачем и как расчитываются эти ошибки, рекомендуем сначала ознакомиться со статьей "error-matrix.html Матрица ошибок и расчет показателей точности тематических карт". Вкратце, эти ошибки являются интегральным показателем точности полученной тематической карты (например классификации). Для подобных расчетов берется 2 слоя, проверочный и проверяемый.

Для начала работы понадобится (вы можете скачать примеры, или использовать свои - похожие):

- ArcGIS с лицензией Spatial Analyst
- Слой границ территории исследования, вектор, lat/long WGS84 (скачать)
- Проверяемый слой, растровая классификация, нужный класс обозначен 1, растр, Albers-Europe (скачать)
- Проверочный слой, вектор, lat/long WGS84 (скачать)
- Таблица в формате dbf, поля ID, RESULT, 4 записи со значениями 1, 2, 3, 4 в поле ID (скачать)

Наша программа будет выполнять набор пространственных операций с векторными и растровыми данными и выдавать в результате значения расчета встречаемости классов в таблицу.

Для начала работы создадим чистый текстовый файл в кодировке UTF-8, назовем его, например, validation.py (или скачаем готовый). В него можно вставлять строки, как они указаны дальше, а потом выполнить его целиком. Альтернативно можно копировать и вставлять их прямо в окно Python и наблюдать за процессом исполнения и появляющимися на диске слоями.

В статье использовался ArcGIS 9.2 build 1450, лицензия ArcInfo.

Подготовка

Начнем с подготовки, любой программе необходимо описание, кто является ее автором и что она делает. Также необходимо указать кодировку, чтобы комментарии на русском не вызывали недовольство интерпретатора языка Python. В целом, этот фрагмент является необязательным. Все комментарии начинаются со знака #, можете также добавить своих.

```
# -*- coding: utf-8 -*-
 # ---------------------------------------------------------------------------
 # validation.py
 # Получение данных для расчета ошибок производителя и пользователя для двух источников
данных V1 и V2
 # Author: Maxim Dubinin (sim@gis-lab.info)
 # ---------------------------------------------------------------------------
```

Дальше, импортируем модуль поддержки Python ArcGIS - arcgisscripting.

```
import arcgisscripting
```

Создадим основной объект геопроцессинга:

```
gp = arcgisscripting.create()
```

и сразу сделаем ему общую настройку, чтобы результаты работы переписывали существующие, наверняка вероятно придется что-то повторять в процессе отладки:

```
gp.overwriteoutput = 1
```

Получаем необходимые лицензии:

```
gp.CheckOutExtension("spatial")
```

Зададим также рабочу папку, где будут храниться исходные и конечные материалы:

```
wd = "D:\\Programming\\Python\\validation\\"
```

Геопроцессинг

Теперь начинается собственно обработка данных. Обычно схема такая, есть некоторая операция, которую нужно выполнить, для нее есть исходный и конечный слой, они задаются прямо перед операцией, для удобства. Общие обозначения, используемые в названиях слоёв:

- sa - территория исследования (район интересов, study area)
- vec - вектор
- ras - растр
- v1 - проверочный набор данных
- v2 - набор данных который проверяют
- dd, alb - lat/long и Albers соответственно.

Исходные данные (sa, v1, v2):

Так как вектор мы обычно храним в lat/long, а растры в проекции Albers, сначала преобразуем вектор в проекцию Albers. Небольшая хитрость, описание системы координат было бы более аккуратно назначить какой-то переменной и потом использовать в gp.Project_management как третий параметр, но почему-то это ArcGIS переварить не может, поэтому использовать длинное описание приходится в команде здесь и далее.

```
sa_dd = wd + "sa_dd.shp"
sa_alb = wd + "sa_alb.shp"
gp.Project_management(sa_dd, sa_alb, "PROJCS'Albers-
Europe',GEOGCS'GCS_Pulkovo_1942',DATUM'D_Pulkovo_1942',SPHEROID'Krasovsky_1940',6378245
.0,298.3,PRIMEM'Greenwich',0.0,UNIT'Degree',0.0174532925199433,PROJECTION'Albers',PARAM
ETER'False_Easting',8500000.0,PARAMETER'False_Northing',0.0,PARAMETER'Central_Meridian'
,45.0,PARAMETER'Standard_Parallel_1',52.0,PARAMETER'Standard_Parallel_2',64.0,PARAMETER
'Latitude_Of_Origin',0.0,UNIT'Meter',1.0", "Pulkovo_1942_To_WGS_1984",
```

```
"GEOGCS'GCS_WGS_1984',DATUM'D_WGS_1984',SPHEROID'WGS_1984',6378137.0,298.257223563,PRIM
EM'Greenwich',0.0,UNIT'Degree',0.0174532925199433")
```

Еще одна небольшая хитрость, в некоторых случаях, если перепроецируемых слой уже где-то открыт (в нашем случае он был открыт в QGIS), операция будет выдавать ошибку, пока вы его не закроете.

Так как у нас теперь есть векторная граница территории исследования, за пределами которой нас ничего не интересует, установим ее как маску. В растровых операциях приводящих к появлению чего-либо за маской, эти значения будут сбрасываться в NODATA.

```
gp.mask = sa_alb
```

Растеризуем границу территории исследования с разрешением таким же, как у растра V2 (он получен на основе Landsat, поэтому 30 метров):

```
sa_ras = wd + "sa_ras"
gp.FeatureToRaster_conversion(sa_alb, "ID", sa_ras, "30")
```

Перепроецируем V1 в ту же систему координат и проекцию:

```
v1_dd = wd + "v1_dd.shp"
v1_alb = wd + "v1_alb.shp"
gp.Project_management(v1_dd, v1_alb, "PROJCS'Albers-
Europe',GEOGCS'GCS_Pulkovo_1942',DATUM'D_Pulkovo_1942',SPHEROID'Krasovsky_1940',6378245
.0,298.3,PRIMEM'Greenwich',0.0,UNIT'Degree',0.0174532925199433,PROJECTION'Albers',PARAM
ETER'False_Easting',8500000.0,PARAMETER'False_Northing',0.0,PARAMETER'Central_Meridian'
,45.0,PARAMETER'Standard_Parallel_1',52.0,PARAMETER'Standard_Parallel_2',64.0,PARAMETER
'Latitude_Of_Origin',0.0,UNIT'Meter',1.0", "Pulkovo_1942_To_WGS_1984",
"GEOGCS'GCS_WGS_1984',DATUM'D_WGS_1984',SPHEROID'WGS_1984',6378137.0,298.257223563,PRIM
EM'Greenwich',0.0,UNIT'Degree',0.0174532925199433")
```

Проверочные данные (V1) немного выходят за границу территории исследования - обрежем их:

```
v1_alb_clip = wd + "v1_alb_clip.shp"
gp.Clip_analysis(v1_alb, sa_alb, v1_alb_clip, "")
```

Растеризуем обрезанный V1:

```
v1_ras = wd + "v1_ras"
gp.FeatureToRaster_conversion(v1_alb_clip, "ID", v1_ras, "30")
```

Переклассифицируем полученный растр, чтобы все значения 1 заменились на 2, а NODATA - на 0.

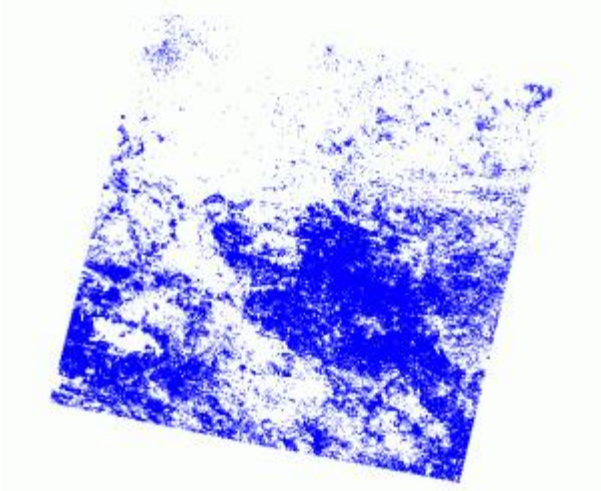
```
v1_ras0 = wd + "v1_ras0"
gp.Reclassify_sa(v1_ras, "VALUE", "1 2;NODATA 0", v1_ras0, "DATA")
```

Складываем V1 и территорию исследования, так как V1 не обязательно захватывает всю территорию, а SA это сплошные нули, то получится растр с таким же охватом как SA.

```
gp.Mosaic_management('"'+v1_ras0+'"', sa_ras, "LAST", "FIRST","", "", "NONE", "0")
```

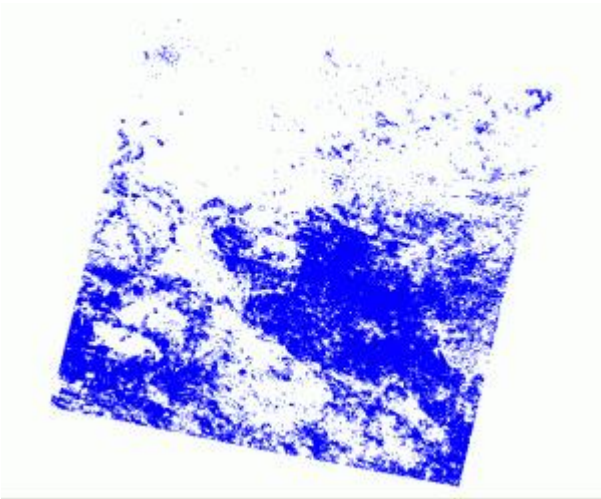
Подготавливаем V2, выбираем все значения равные 1, это класс который нам нужно проверить:

```
v2_ras = wd + "v2"
con1 = wd + "con1"
gp.Con_sa(v2_ras, 1, con1, 0, "\"VALUE\" = 1")
```



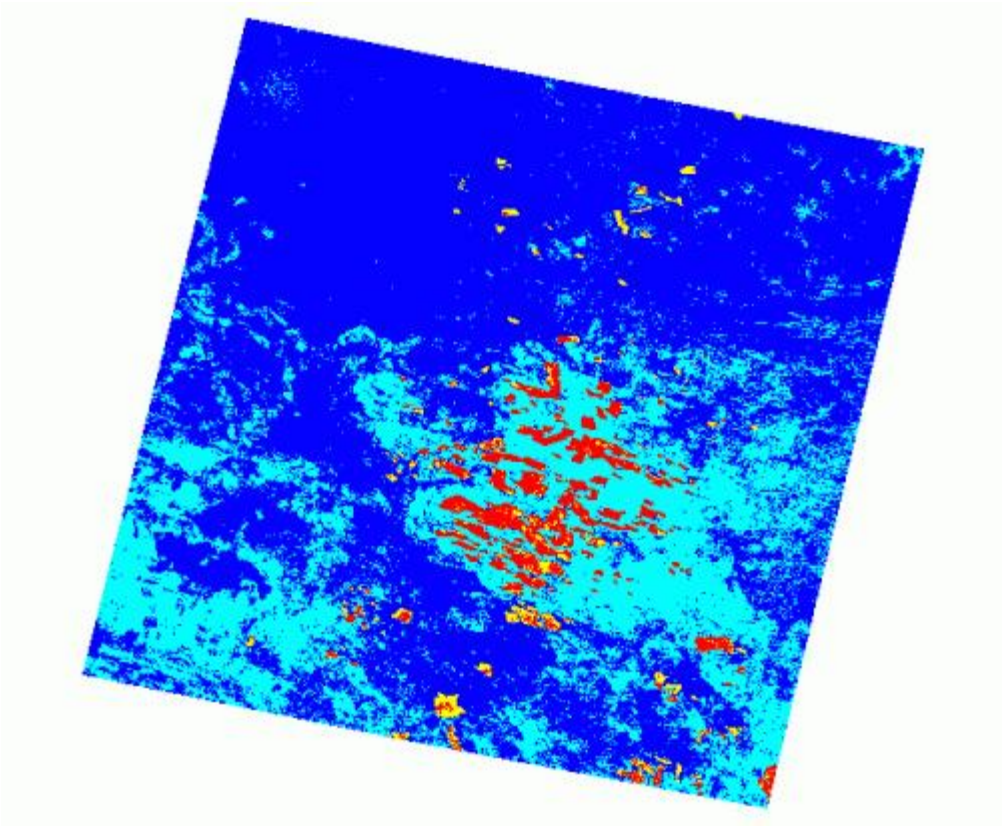
Сглаживаем результат фильтром 3x3:

```
v2_f = wd + "v2_f"
gp.FocalStatistics_sa(con1, v2_f, "Circle 3 CELL", "MAJORITY", "DATA")
```



Суммируем V1 и V2, в результате значение 3 получится у территорий которые есть и на V1 и на V2, значение 2 - только на V1, 1 - только на V2, 0 - ни там ни там.

```
res = wd + "res"
gp.SingleOutputMapAlgebra_sa("sa_ras + v2_f", res)
```



Это наш результат (3 - красный, 2 - желтый, 1 - голубой, 0 - синий), в принципе, все необходимые для расчетов значения можно извлечь из таблицы грида, показывающую сколько пикселей какого класса получилось, в целях иллюстрации пойдем по несколько другому пути и получим значения в таблицу.

Для этого, сделаем вид из готовой таблицы, чтобы объединить ее с таблице результата:

```
result_dbf = wd + "result.dbf"
result_view = "result_view"
gp.MakeTableView_management(result_dbf, result_view)
```

Объединяем таблицы по полям ID и VALUE:

```
gp.AddJoin_management(result_view, "ID", res, "VALUE", "KEEP_ALL")
```

Переносим значения:

```
gp.CalculateField_management(result_view, "RESULT", "" + resvat +":COUNT", "VB")
```

Результат выглядит примерно таким образом.

ID	RESULT
0	23169981
1	10272529
2	393835
3	1138037

Используя эти данные легко посчитать например точность производителя:

1138037/(1138037+393835) = 0.742906 = 74%

и пользователя:

1138037/(1138037+10272529) = 0.099735 = 9.9%

Действительно, как видно из рисунка выше, 3/4 территорий которые должны были попасть в нужный класс V2 - в него попали (класс 3), 1/4 - не попала (класс 2), однако в него же попало и огромное количество лишнего (класс 1).

Подробнее о расчете точности и интерпретации можно прочитать в error-matrix.html специальной статье.

Осталось только прибраться за собой, отсоединить таблицу и удалить промежуточные результаты:

```
gp.RemoveJoin_management(result_view, resvat)
gp.Delete_management(sa_ras, "")
gp.Delete_management(sa_alb, "")
gp.Delete_management(v1_alb, "")
gp.Delete_management(v1_alb_clip, "")
gp.Delete_management(v1_ras, "")
gp.Delete_management(v1_ras0, "")
gp.Delete_management(con1, "")
gp.Delete_management(v2_f, "")
```

Обработка программных ошибок

По умолчанию, если вы ошиблись в вводе параметров для одной из операций геопроцессинга, вы будете получать маловнятные сообщения об ошибках типа такого:

```
Traceback (most recent call last):
  File "", line 1, in
  File "", line 2, in Select_analysis
pywintypes.com_error: (-2147467259, 'Unspecified error', None, None)
```

Для получения более содержательного сообщения, необходимо "завернуть" вызов команды выдающей ошибку в блок try: except pywintypes.com_error, вот таким образом:

```
try:
        wrong_command_here
except pywintypes.com_error:
        # Получить сообщение об ошибке от геопроцессора
        msgs = gp.GetMessage(0)

        # Return gp error messages for use with a script tool
        gp.AddError(msgs)

        # Вывести ошибки для использования в Python/PythonWin
        print msgs
```

Для использования этой возможности необходимо установить Python for Windows extensions для соответствующей версии Python. Если запускать строку вызывающую ошибку в этом блоке, информации будет чуть больше (хотя тоже не всегда достаточно).

Скачать готовый скрипт.

Ссылки по теме

- Матрица ошибок и расчет показателей точности тематических карт

Обсудить в форуме Комментариев — 0

Последнее обновление: 2014-05-15 01:37

Дата создания: 05.06.2010

Автор(ы): Максим Дубинин