GeoPandas: Spatial join

Dr. Cornelis Stal

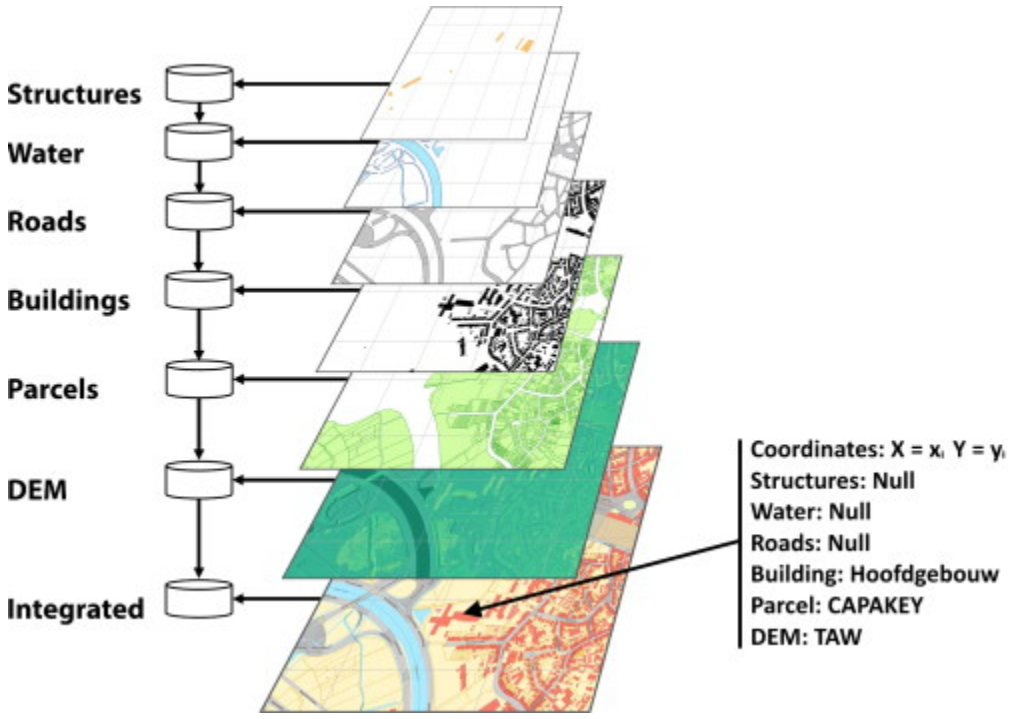
April 28, 2022

1 Spatial joins

Een *spatial join* (vrij vertaald als *ruimtelijke koppeling* maakt gebruik van binaire predicaten zoals `intersects` en `crosses` om twee `GeoDataFrames` met elkaar te koppelen op basis van een bepaalde ruimtelijke relatie tussen de verschillende geometriën. Zoals bij attribuutkoppelingen zullen we twee datasets, die niet noodzakelijkerwijze met dezelfde doelstelling aangemaakt zijn, dus met elkaar combineren op basis van bepaalde kenmerken. In tegenstelling tot de attribuutkoppelingen maken we echter geen gebruik van gemeenschappelijke velden, maar van ruimtelijek condities.

Spatial joins worden bijvoorbeeeld veel gebruikt om attributen van een polygoon over te dragen naar puntobjecten. Zoals we zodadelijk in onderstaande voorbeeld zullen demonstreren, wordt de koppeling gemaakt voor ieder punt gelegen binnen een bepaalde polygoon.

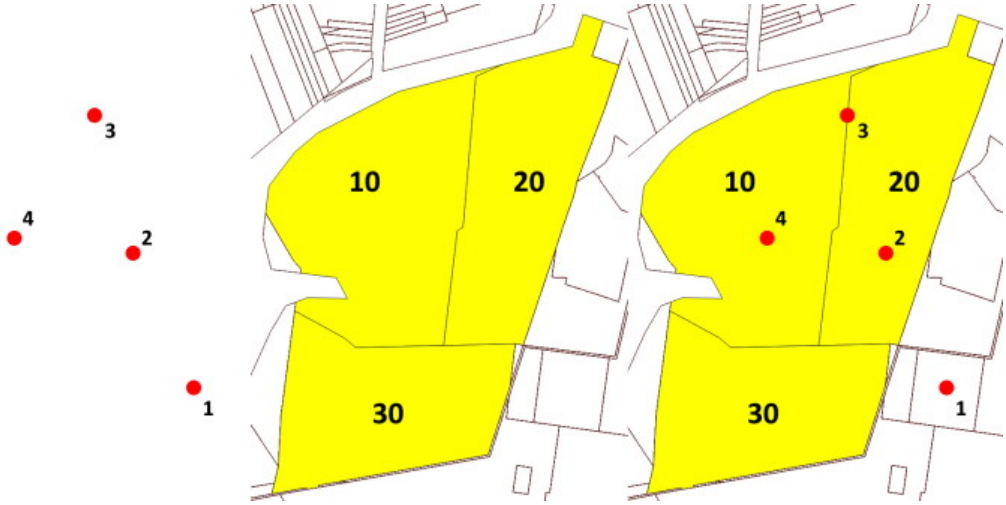
In onderstaande figuur illustreren we hoe we waarde kunnen toegoeven aan individuele lagen door deze te koppelen. Bij een *spatial join* zal dit telkens koppel per koppel gebeuren.



Deze tutorial is een vertaling van de 'example guide' op https://geopandas.org.

1.1 Verschillende type ruimtelijke relaties

`GeoPandas` ondersteunt momenteel verschillende typen *spatial joins*. In onderstaand voorbeeld illustreren we verschillende concepten aan de hand van een hypothetisch voorbeeld, waarbij we beschikken over twee datasets, namelijk een set met punten en een set met polygonen:



tabel `pts`

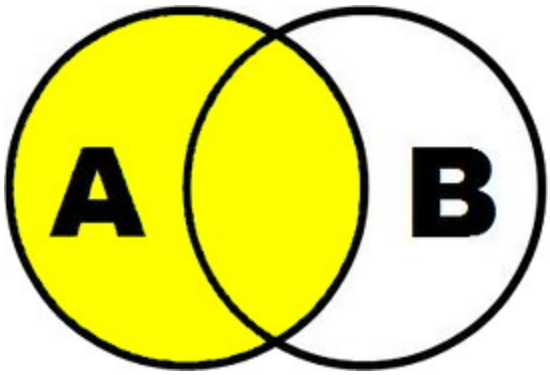
geom	id
POINT(x_1, y_1)	1
POINT(x_2, y_2)	2
POINT(x_3, y_3)	3
POINT(x_4, y_4)	4

tabel `polys`

geom	id
POLYGON(($x_{10,1}$, $y_{10,1}$), … , ($x_{10,n}$, $y_{10,n}$))	10
POLYGON(($x_{20,1}$, $y_{20,1}$), … , ($x_{20,n}$, $y_{20,n}$))	20
POLYGON(($x_{30,1}$, $y_{30,1}$), … , ($x_{30,n}$, $y_{30,n}$))	30

1.1.1 *Left outer join*

Bij een *LEFT OUTER JOIN* (`how='left'`) behouden we **alle** rijen uit de linker tabel. Indien nodig dupliceren we deze rijen wanneer er meerdere overeenkosten gevonden worden in de rechter tabel. We behouden attributen uit de rechter tabel indien deze intersecten met de linker tabel en laten alle andere velden achter wegen. Een *LEFT OUTER JOIN* impliceert dat we dus enkel geïnteresseerd zijn in het behoud van de geometriën uit de linker tabel.



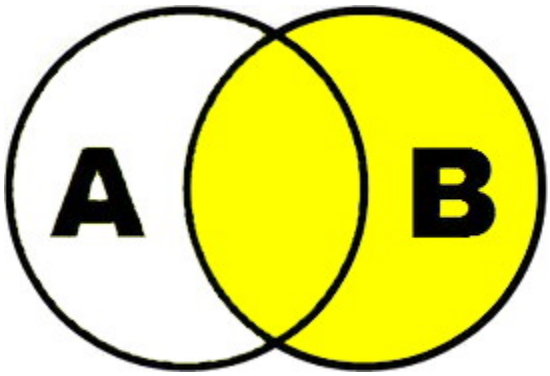
Zonder nu al vooruit te lopen op de codering in `GeoPandas` geven we hieronder de equivalente *query* voor een *LEFT OUTER JOIN* in PostGIS:

```
SELECT pts.geom, pts.id as ptid, polys.id as polyid
FROM pts
LEFT OUTER JOIN polys
ON ST_Intersects(pts.geom, polys.geom);

      geom     | ptid | polyid
---------------+------+--------
 POINT(x4, y4) |    4 |     10
 POINT(x3, y3) |    3 |     10
 POINT(x3, y3) |    3 |     20
 POINT(x2, y2) |    2 |     20
 POINT(x1, y1) |    1 |
(5 rows)
```

1.1.2 *Right outer join*

Bij een *RIGHT OUTER JOIN* (`how='right'`) behouden we **alle** rijen uit de rechter tabel. Indien nodig dupliceren we deze rijen wanneer er meerdere overeenkosten gevonden worden in de linker tabel. We behouden attributen uit de linker tabel indien deze intersecten met de rechter tabel en laten alle andere velden achter wegen. Een *RIGHT OUTER JOIN* impliceert dat we dus enkel geïnteresseerd zijn in het behoud van de geometriën uit de rechter tabel.



Zonder nu al vooruit te lopen op de codering in `GeoPandas` geven we hieronder de equivalente *query*

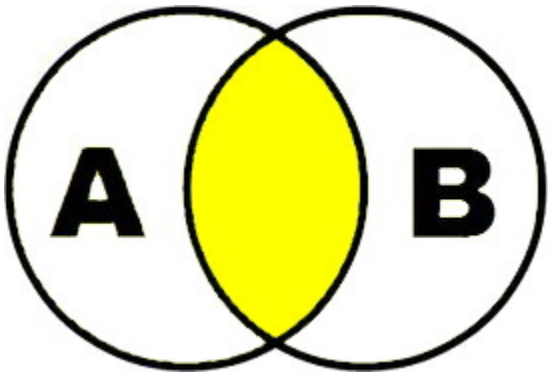
voor een *RIGHT OUTER JOIN* in PostGIS:

```
SELECT polys.geom, pts.id as ptid, polys.id as polyid
FROM pts
RIGHT OUTER JOIN polys
ON ST_Intersects(pts.geom, polys.geom);
```

```
                      geom                       | ptid | polyid
-------------------------------------------------+------+--------
 POLYGON((x10,1, y10,1), ... , (x10,n, y10,n)) |    4 |     10
 POLYGON((x10,1, y10,1), ... , (x10,n, y10,n)) |    3 |     10
 POLYGON((x20,1, y20,1), ... , (x20,n, y20,n)) |    3 |     20
 POLYGON((x20,1, y20,1), ... , (x20,n, y20,n)) |    2 |     20
 POLYGON((x30,1, y30,1), ... , (x30,n, y30,n)) |      |     30
(5 rows)
```

1.1.3 *Inner join*

Bij een *INNER JOIN* (`how='inner'`) behouden we enkel de rijen uit de linker- en rechter tabel wanneer het binaire predicaat gelijk is aan `True`. Wanneer er meerdere correspondenties terug te vinden zijn in een van beide tabellen, worden entiteiten gedupliceerd. Alle attirbuten uit de linker- en rechter tabel worden behouden als ze intersecten en alle andere rijen verdwijnen.



Zonder nu al vooruit te lopen op de codering in GeoPandas geven we hieronder de equivalente query voor een *INNER JOIN* in PostGIS:

```
SELECT pts.geom, pts.id as ptid, polys.id as polyid
FROM pts
INNER JOIN polys
ON ST_Intersects(pts.geom, polys.geom);
```

```
      geom      | ptid | polyid
----------------+------+--------
 POINT(x4, y4) |    4 |     10
 POINT(x3, y3) |    3 |     10
 POINT(x3, y3) |    3 |     20
 POINT(x2, y2) |    2 |     20
(4 rows)
```

1.2 *Spatial joins* tussen twee *GeoDataFrames*

In onderstaande voorbeeld zullen we de implementatie van *spatial joins* in `GeoPandas` illustreren aan de hand van de koppeling van administratieve percelen uit het Vlaamse GRB via WFS aan de adrespunten uit het Vlaamse CRAB uit het WFS. Beide datasets zullen we eerst downloaden met `requests` en verwerken tot twee afzonderlijke `GeoDataFrames`. Voor deze demo maken we gebruik van de deelgemeente Kinrooi, waarbij de CAPAKEY van de daar gelegen percelen starten met `72018` (deelgemeente Kinrooi). We starten met het downloaden van de administratieve percelen:

[ ]:
```
import requests
import matplotlib.pyplot as plt
import geopandas as gpd
# Instelling om figuren wat groter af te beelden
plt.rcParams['figure.figsize'] = [15, 5]

# WFS endpoint
urlADP = 'https://geoservices.informatievlaanderen.be/overdrachtdiensten/GRB/
 ↪wfs'
# Parameters
paramsADP = {
    "REQUEST": "GetFeature",
    "SERVICE": "WFS",
    "SRSNAME": "urn:ogc:def:crs:EPSG::31370",
    "STARTINDEX": "0",
    "OUTPUTFORMAT": "application/json",
    "TYPENAMES": "GRB:ADP",
    "VERSION": "2.0.0",
    "cql_filter": "CAPAKEY LIKE'72018%'"}
rADP = requests.get(urlADP, params = paramsADP)
gdfADP = gpd.read_file(rADP.text)
gdfADP.head()
```

Vervolgens downloaden we de adrespunten voor Kinrooi:

[ ]:
```
# WFS endpoint
urlCRAB = "https://geoservices.informatievlaanderen.be/overdrachtdiensten/
 ↪Adressen/wfs"
# Parameters
paramsCRAB = {
    "REQUEST": "GetFeature",
    "SERVICE": "WFS",
    "SRSNAME": "urn:ogc:def:crs:EPSG::31370",
    "STARTINDEX": "0",
    "TYPENAMES": "Adressen:Adrespos",
    "OUTPUTFORMAT": "application/json",
    "VERSION": "2.0.0",
    "cql_filter": "nisgemeentecode='72018'"
}
```

```
rCRAB = requests.get(urlCRAB, paramsCRAB)
gdfCRAB = gpd.read_file(rCRAB.text)
gdfCRAB.head()
```

Uiteraard kunnen we beide datasets ook grafisch voorstellen:

[ ]:
```
ax = gdfADP.plot(edgecolor='#B2DF8A', facecolor='#DAFAB5' )
gdfCRAB.plot(ax=ax, color="gray", markersize=1)
```

1.3 Joins

Met `GeoPandas` voeren we een een *spatial join* uit met de `sjoin`-methode. Meer informatie kan hier worden teruggevonden. We starten met een *LEFT OUTER JOIN*:

[ ]:
```
join_left_df = gpd.sjoin(gdfCRAB, gdfADP, how="left")
join_left_df.head()
```

In bovenstaand resultaat zien we voor ieder adres in het CRAB de gegevens van 0 of 1 administratistratieve percelen. Theoretisch gezien zou een punt kunnen resulteren in meerdere percelen (i.e. wanneer een adres exact gelegen is op een perceelsgrens), maar deze situatie wordt hier buiten beschouwing gelaten. Wanneer er voor een adres corresponderende perceelsgegevens beschikbaar zijn, worden deze weergegeven in de corresponderende kolomen. Als dit niet het geval is, zoals wanneer een adres gelegen is buiten de deelgemeente Kinrooi, zal een `NaN`-waarde ingegeven worden.

> **Opmerking:** in onderstaand voorbeeld gebruiken we `sjoin` als een algemene methode van de `GeoPandas`-klasse. In de oorsprokelijke reeks tutorials, die hier terug te vinden zijn, wordt `sjoin` als een methode van een `GeoDataFrame` toegepast:
>
> `join_left_df = gdfCRAB.sjoin(gdfADP, how="left")`
>
> Beide methoden resulteren (voorlopig) in dezelfde output.

De 'RIGHT OUTER JOIN' werkt op een soortgelijke manier en met deze data resulteert dit in een verzameling administratieve percelen. Voor ieder perceel waarin een adrespunt gelegen is, worden de attributen uit het CRAB overgedragen. Indien er binnen een perceel meerdere adrespunten geleven zijn, worden deze percelen gedupliceerd:

[ ]:
```
join_right_df = gdfCRAB.sjoin(gdfADP, how="right")
join_right_df.head()
```

Tot slot berekenen we de *INNER JOIN* op beide datasets. Als we opnieuw de adrespunten links plaatsen en de percelen rechts, krijgen we ieder adres een of meerdere keren terug waarvoor corresponderende percelen beschikbaar zijn. Merk op dat hierbij geen `NaN`-waarden verschijnen:

[ ]:
```
join_inner_df = gpd.sjoin(gdfCRAB, gdfADP, how="inner")
join_inner_df.head()
```

Zoals we in de handleiding over `sjoin` kunnen lezen, hoeven we ons niet te beperken tot predicaten van het type `intersection`. Hoewel dit de standaard relatie bepaald, kunnen we iedere

geometrische methode toepassen die ondersteund wordt door `Shapely`, zoals omschreven in de corresponderende documentatie. Voorbeelden zijn `within`, `touches`, `overlaps`, … Vanzelfsprekend passen we deze methodes telkens toe met de juiste waarde voor het `how`-veld:

[ ]:
```
join_inner_df = gpd.sjoin(gdfCRAB, gdfADP, how="inner", predicate="within")
join_inner_df
```