Final_Notebook

August 1, 2020

1 Capstone Project - The Battle of Neighborhoods (Week 1)

Table of Contents

1.1 Introduction

The objective of this project is to create a way for a Real State Agency to select the best region of Greater London to offer their clients. The idea is to determine the regions with the greater concentration of venues of each category that can be found in Foursquare (which are 'Arts & Entertainment', 'College & University', 'Food', 'Nightlife Spot', 'Outdoors & Recreation', 'Professional & Other Places', 'Residence', 'Shop & Service', and 'Travel & Transport'). This way, the agency can adapt their offers of residence to rent to match the interests of specific clients, depending on their priorities.

Let's say, for example, that a couple wants to rent a flat in Greater London, and that this couple greatly values visits to museums and the like as a form of leisure. With knowledge about the regions where "Art & Entertainment" venues are more concentrated, the agency can suggest flats in these regions for the couple.

In a similar way, a client might value spending their leisure time in parks and other outdoor venues. Then the Real State Agency can accommodate his desire by suggesting a flat located in a region of high concentration of "Outdoors & Recreation" venues.

So the stakeholders here are Real State Agencies in the area of Greater London, and the project will provide information to them that can help them better accommodate their clients' needs.

1.2 Data

1.2.1 Defining the data to be employed

Information about all points of interest (hereafter POI) in the Greater London will be collected from Foursquare using its Places API (https://enterprise.foursquare.com/products/places), for which I created a free developer account ("Personal" tier). This account allows for 99,500 regular calls per day, more than enough to collect the necessary data. Since this data will be relatively bulky, after retrieval from the Foursquare's database, it will be persisted to my local drive, so that analysis can ensue without the need to download everything again, should something wrong happen.

In order to check if the location of the POI is inside the territory of the Greater London, it should suffice to check the city name in the Foursquare data. However, I found that this particular bit of information is relatively inconsistent in the Foursquare database. The city name is often incorrect, or even absent (at least for the area of the Greater London). So I decided to use a more robust, though slightly more convoluted, approach. This is composed of 1 to 4 steps, as follows:

1. Using the POI latitude and longitude as input parameters, its postal code will be obtained from the Bing Maps REST Services (https://docs.microsoft.com/en-us/bingmaps/rest-services/). For this purpose, a "Basic/Dev/Test" account was created. Then, with the postal code as input, the name of the city in which the POI is located will be obtained using Postcode.io (http://api.postcodes.io/) - this service does not demand the use of a key, so no account needs to be created.

2. If a valid postal code (2 sequences of 3 alphanumeric characters, separated by a space) cannot be fetched from Bing, the algorithm will verify if a valid postal code is available in the Foursquare data. If yes, Postcode.io will be consulted using the Foursquare postal code.

3. If no postal code can be obtained from either Bing or Foursquare, the algorithm will try to fetch the city name directly from the Foursquare data.

4. Finally, if step 3 fails, the POI is dropped.

The POI will be kept if its city name is in the list of the 32 Greater London boroughs. This list will be taken from the first table of https://en.wikipedia.org/wiki/London_boroughs. Please, note that although the correct denomination of the Greater London administrative divisons is borough, this data can be found in different databases under diverse labels. In Foursquare, it is called (when present...) "city". In Postcode.io it is labeled "admin_district".

The data that will be used for determining the regions of high concentration of venues for each category is the location of each venue (latitude and longitude), and the category to which it pertains. Analysis will be mainly composed of Density-Based clustering of the venues, grouped by category.

1.2.2 Getting and cleaning the data

[97]:
```
import requests
import pandas as pd
import shapely.geometry
from pyproj import Proj
import math
import folium
import requests
```

```
import pickle
```

[98]:
```
FOURSQUARE_CLIENT_ID = 'B4VGNB4EAPO2VA2GPPBPARNJOGW4LE1SAV4YNH4G4KAROIFG' #␣
↪your Foursquare ID
FOURSQUARE_CLIENT_SECRET = 'FG5GCAFKU3E3MSSG25QEU4YNVSRYNPZDMXQFEIJZBNZ52S50' #␣
↪your Foursquare Secret
FOURSQUARE_VERSION = '20180605' # Foursquare API version
```

[99]:
```
BING_KEY = 'Atm4bmL7hptbxYV1-SAg8mrF7RnSdpAPpB5Xj9A5-RNxS42nEksJ38OYGnS0wO4h'
```

[100]:
```
def lonlat_to_xy(lon, lat):
    p = Proj(proj='utm',zone=33,ellps='WGS84', preserve_units=False)
    xy = p(lon, lat)
    return xy[0], xy[1]

def xy_to_lonlat(x, y):
    p = Proj(proj='utm', zone=33, ellps='WGS84', preserve_units=False)
    lonlat = p(x, y, inverse=True)
    return lonlat[0], lonlat[1]

def calc_xy_distance(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    return math.sqrt(dx*dx + dy*dy)

london_center=[51.4914995,-0.0889]
x, y = lonlat_to_xy(london_center[1], london_center[0])
print('London center UTM X={}, Y={}'.format(x, y))
lo, la = xy_to_lonlat(x, y)
print('London center longitude={}, latitude={}'.format(lo, la))
```

```
London center UTM X=-544714.2265471916, Y=5813248.535185799
London center longitude=-0.08889999999999917, latitude=51.49149950000001
```

[101]:
```
london_center_x, london_center_y = lonlat_to_xy(london_center[1],␣
↪london_center[0]) # City center in Cartesian coordinates

k = math.sqrt(3) / 2 # Vertical offset for hexagonal grid cells
x_min = london_center_x - 36000
x_step = 1200
y_min = london_center_y - 36000 - (int(126/k)*k*3600 - 72000)/2
y_step = 1200 * k

latitudes = []
longitudes = []
distances_from_center = []
xs = []
```

```
ys = []
for i in range(0, int(378/k)):
    y = y_min + i * y_step
    x_offset = 1800 if i%2==0 else 0
    for j in range(0, 378):
        x = x_min + j * x_step + x_offset
        distance_from_center = calc_xy_distance(london_center_x,␣
 ↪london_center_y, x, y)
        if (distance_from_center <= 36001):
            lon, lat = xy_to_lonlat(x, y)
            latitudes.append(lat)
            longitudes.append(lon)
            distances_from_center.append(distance_from_center)
            xs.append(x)
            ys.append(y)

print(len(latitudes), 'candidate neighborhood centers generated.')
```

```
3257 candidate neighborhood centers generated.
```

[104]:
```
map_london = folium.Map(location=london_center, zoom_start=13)
folium.Marker(london_center, popup='Greater London').add_to(map_london)
for lat, lon in zip(latitudes, longitudes):
    folium.Circle([lat, lon], radius=600, color='blue', fill=False).
 ↪add_to(map_london)
map_london
```

[104]:
```
<folium.folium.Map at 0x7f7b09516f70>
```

[105]:
```
boroughs = pd.read_html("https://en.wikipedia.org/wiki/London_boroughs")
```

[106]:
```
boroughs = boroughs[2]['London borough'].str.replace('(\[(.*?)\])','').str.
 ↪lower()
boroughs
```

[106]:
```
0                    camden
1                 greenwich
2                   hackney
3             hammersmith
4                 islington
5     kensington and chelsea
6                   lambeth
7                  lewisham
8                 southwark
9             tower hamlets
10              wandsworth
11              westminster
```

```
12                   barking
13                    barnet
14                    bexley
15                     brent
16                   bromley
17                   croydon
18                    ealing
19                   enfield
20                  haringey
21                    harrow
22                  havering
23                hillingdon
24                  hounslow
25      kingston upon thames
26                    merton
27                    newham
28                 redbridge
29      richmond upon thames
30                    sutton
31            waltham forest
Name: London borough, dtype: object
```

[107]:
```
url = 'https://api.foursquare.com/v2/venues/categories?
 ↪client_id={}&client_secret={}&v={}'.format(
            FOURSQUARE_CLIENT_ID,
            FOURSQUARE_CLIENT_SECRET,
            FOURSQUARE_VERSION)

categories = requests.get(url).json()
```

Get the most up-to-date list of categories, with their names and ids

[108]:
```
categories_names = [item['name'] for item in␣
 ↪categories['response']['categories']]
categories_ids = [item['id'] for item in categories['response']['categories']]
```

[109]:
```
categories_names
```

[109]:
```
['Arts & Entertainment',
 'College & University',
 'Event',
 'Food',
 'Nightlife Spot',
 'Outdoors & Recreation',
 'Professional & Other Places',
 'Residence',
 'Shop & Service',
```

```
 'Travel & Transport']
```

[ ]:
```
radius = 600
limit = 50
venues={}

for cat in categories_names:
    venues[cat] = []

limits = range(0,3251,50)

for i in range(65):

    for counter in range(limits[i], limits[i+1]):

        print(counter)

        for cat_name, cat_id in zip(categories_names, categories_ids):
            url = 'https://api.foursquare.com/v2/venues/explore?
↪client_id={}&client_secret={}&v={}&ll={},{}&categoryId={}&radius={}&limit={}'.
↪format(FOURSQUARE_CLIENT_ID, FOURSQUARE_CLIENT_SECRET, FOURSQUARE_VERSION,␣
↪latitudes[counter], longitudes[counter],cat_id, radius, limit)
            results = requests.get(url).json()
            try:
                for item in results['response']['groups'][0]['items']:
                    found_postcode = False
                    url_bing = 'http://dev.virtualearth.net/REST/v1/Locations/
↪{},{}?key={}'.
↪format(item['venue']['location']['lat'],item['venue']['location']['lng'],BING_KEY)
                    result_bing = requests.get(url_bing).json()
                    postcode =␣
↪result_bing['resourceSets'][0]['resources'][0]['address']['postalCode']
                    if (len(postcode) == 7):
                        found_postcode = True

                    if (found_postcode == False):
                        postcode= item['venue']['location']['postalCode']
                        if (len(postcode) == 7):
                            found_postcode = True

                    if (found_postcode == True):
                        url = 'https://api.postcodes.io/postcodes/{}'.
↪format(postcode)
                        place = requests.get(url).json()
                        admin_district = place['result']['admin_district']
                    else:
                        admin_district = item['venue']['location']['city']
```

```
                    if admin_district.lower() not in boroughs.values:
                        continue
                    venues[cat_name].
↪append([counter,admin_district,postcode,item['venue']['name'],␣
↪item['venue']['location']['lat'], item['venue']['location']['lng']])
            except:
                continue
    filename = 'dict_' + str(limits[i+1]) + '.pickle'
    with open(filename, 'wb') as handle:
        pickle.dump(venues, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

[110]:
```
with open('/home/eduardo/Documents/DATA_SCIENCE/IBM_DATA_SCIENCE_CAPSTONE/
↪DADOS_CAPSTONE_COMPLETOS.pickle', 'rb') as handle:
    venues = pickle.load(handle)
```

1.3 Methodology

1.4 Results

1.5 Discussion

1.6 Conclusion

1.7 Acknowledgements

This project was inspired by an example presented in the 5th week of the Applied Data Science Capstone web page at https://cocl.us/coursera_capstone_notebook.

Particularly, part of the code used to divide the area of interest in small circular regions in an hexagonal arrangement was used.

However, I believe that this inspiration, and use of part of its code, should not be regarded as plagiarism because I do many things differently:

- The purpose of both projects is similar, but not quite the same.
- I updated the part of the code that was copied from the other project to use version 2 of the pyproj module (https://pypi.org/project/pyproj/), as the use of version 1 functions issues deprecation warnings.
- I use other APIs that the inspiring project do not use, along with the Foursquare one.
- I query the Foursquare database and preprocess the results in a different way.
- My project is more encompassing, as I do not analyze only restaurants, but almost all Foursquare venue categories (except “Event”).
- I employ different clustering algorithms.

[ ]: