P3: Wrangle OpenStreetMap Data

Map area: Gastein valley, Austria

06.05.2016

Oleg Leyzerov

Overview

Third project of the Udacity Data Analyst Nanodegree covers the aspects of the data wrangling: gathering, extracting, cleaning and storing the data. I was supposed to choose the region and wrangle the data set of the OpenStreetMap. I've chosen the place where I live - Gastein valley - one of the most popular tourist destinations in Austria, which is

famous for the great skiing area and thermal water. In my project I'm going to concentrate on the accommodation data (hotels, apartments, etc.) as it's always important for the travel industry.

Content

1. Problems encountered in your map

After downloading the data of the Gastein valley and I've run a number of explorational functions in my python program to identify which tag's keys and values are used to code accommodations, their address and other contact information. While there was no much rubbish in the data, I noticed three main problems with the data, which I will discuss in the following order:

Different writing of the street names (using "-" and spaces)

Part of the accommodations don't have the address information

Phone number is written always differently

Different writing of the street names

There are different ways how to harmonize street writing: we might use the language rules or the official list issued by the government (if exists). In German language, the rules of writing the street names are quite complicated. So, I've found the catalog of streets[1], issued by the government and use it as a standard. I'm comparing the streets in osm file with the standard writing and, if the writing is different, I'm searching the replacement using get_close_matches function from difflib library.

```
def get_street_std_list():
    with open(street_std_file,'rb') as f:
        addr = csv.DictReader(f,delimiter=';')
        for elem in addr:
            street_std_list.append(elem['Straßenname'.decode('utf-8')].encode('utf-8'))
```

```
def street_std(el):
    if len(street_std_list) == 0:
        get_street_std_list()
    if el and 'address' in el and 'street' in el['address'] and el['address']['street'].encode('utf-8') not in
street_std_list:
        el['address']['street'] = get_close_matches(el['address']['street'].encode('utf-8'), street_std_list,
n=1)[0].decode('utf-8')
    return el
```

Part of the accommodations don't have the address information

The most frustrating thing for me, if I decide to build the catalog of accommodations, is the fact that such an important information like address is missing in some of the cases. So, I've decided to use Google Places API to extend the information about my accommodations.

I'm using two iterations to get the necessary information: first I'm searching for the place using latitude and longitude information which is always available in OSM data. Even if the accommodation is coded with "way" object but not "node", I'm using the coordinates of the referenced node. As a response, I'm receiving place_id, which I can use on the second step to get address.

```
def google_adr(el):
    API_KEY = <api_key>
            url =
        'https://maps.googleapis.com/maps/api/place/nearbysearch/json?location=%f,%f&radius=25&key='%(e
        l["pos"][0],el["pos"][1]) + API_KEY
            r = requests.get(url)
            if r.status_code == requests.codes.ok:
                for response in r.json()['results']:
                    if "lodging" in response['types']:
                        place_id = response['place_id']
                        url2 =
        'https://maps.googleapis.com/maps/api/place/details/json?placeid=%s&key='%(place_id)+API_KEY
                        r2 = requests.get(url2)
                        if r2.status_code == requests.codes.ok:
                            if 'result' in r2.json():
                                el['address']={}
                                for elem in r2.json()['result']['address_components']:
```

```
                    if elem['types'][0] == 'street_number':
                        el['address']['housenumber'] = elem['long_name']
                    if elem['types'][0] == 'route':
                        el['address']['street'] = elem['long_name']
                    if elem['types'][0] == 'locality':
                        el['address']['city'] = elem['long_name']
                    if elem['types'][0] == 'country':
                        el['address']['country'] = elem['short_name']
                    if elem['types'][0] == 'postal_code':
                        el['address']['postcode'] = elem['short_name']
                el = street_std(el)
                el = postcode_audit(el)
            else:
                print "error"
            return el
```

After getting the information, again I'm comparing the street with the standard writing.

Google address data has some issues, f.ex. With postcodes format (f.ex. 5640 vs. A-5640). So I've written the small function using the regular expression to elaborate this:

```
def postcode_audit(el):
    if 'postcode' in el['address']:
        if len(el['address']['postcode'])>4:
            m = re.search(r'\d\d\d\d$',el['address']['postcode'])
            if m:
                el['address']['postcode'] = m.group()
    return el
```

Phone number is written always differently

Phone numbers are written quite different. So I've just decided to use single format for all of them. I've realized it using the sequence of regular expressions and re.sub function.

```
def phone_audit(el):
    if "phone" in el:
        el['phone'] = re.sub(r'^00','+',el['phone'])
        el['phone'] = re.sub(r'\(0\)','',el['phone'])
        el['phone'] = '+'+re.sub(r'[ -//]','',el['phone'])
        el['phone'] = re.sub(r'^\+430',"+43",el['phone'])
```

```
    if "fax" in el:
        el['fax'] = re.sub(r'^00','+',el['phone'])
```

2. Overview of the data

The osm file size is around 50MB.

Below are the statistics of the data I've used in the project:

1. Unique users:

```
def unique_users(filename):
    users = set()
    for _, element in ET.iterparse(filename):
        if 'uid' in element.attrib:
            users.add(element.attrib['uid'])
    return users
def main():
            users = unique_users(filename):
            print len(users)
>>235
```

2. Number of tags

```
def audit_tag(filename):
            cnt = Counter()
            for event, elem in ET.iterparse(filename, events=("start",)):
                        cnt[elem.tag] += 1
                        elem.clear()
            print_sorted_dict(cnt)
```

Result:

nd 282033, node 244588, tag 74900, way 18160, member 12949, relation 283, osm 1, note 1, meta 1, bounds 1

3. To choose the accommodations, I'm using the tag key tourism and values ['hotel', 'chalet', 'guest_house', 'apartment', 'hostel'].

Below are results for "node" and "way":

Counter({'information': 468, 'viewpoint': 19, 'hotel': 13, 'artwork': 12, 'picnic_site': 8, 'chalet': 4, 'alpine_hut': 4, 'guest_house': 3, 'museum': 1, 'apartment': 1, 'camp_site': 1})

Counter({'alpine_hut': 60, 'hotel': 50, 'guest_house': 13, 'chalet': 5, 'apartment': 4, 'museum': 3, 'picnic_site': 2, 'information': 1, 'camp_site': 1, 'hostel': 1})

Here are also some MongoDB queries to obtain the statistics:

Most popular city

```
pipeline = [{"$group" : {"_id" : "$address.city", "count" : {"$sum" : 1}}},
            {"$sort" : {"count" : -1}},
            {"$limit" : 1}]
Result: [{u'_id': u'Bad Gastein', u'count': 42}]
```

2 most populars postcodes:

```
pipeline = [{"$group" : {"_id" : "$address.postcode", "count" : {"$sum" : 1}}},
            {"$sort" : {"count" : -1}},
            {"$limit" : 2}]
Resutl: [{u'_id': u'5640', u'count': 32}, {u'_id': u'5630', u'count': 30}]
# Number of documents (accommodations)
> db.char.find().count()
98
# Number of nodes
db.char.find({"type":"node"}).count()
21
# Number of ways
db.char.find({"type":"way"}).count()
77
```

3. Other ideas about the datasets

As a hotel manager, I know exactly how important for the tourists to have accurate data about the accommodations. The travel agencies must always support the customers with the actual contact information. From the other side, companies who are doing business with the hotels might also win from the information about the potential customers. F.ex. I've used to implement WiFi solutions at the hotels and used hotel's wifi scores from booking.com to target potential customers.

I like the opportunity Google Places API gave me to improve my data. I would work further with Google to extend contact information as well as crawl properties' web pages to get more contact information (like facebook page) and some online travel agencies to get more information if the accommodation is open or not.

Adding more information from more sources bring more issues with cleaning the data. Even the case with Google and addresses was a great example of how clean the data was in OSM (clean but limited). I should write additional functions to clean google's data as well (like post code)

References

1. Statistik Austria:
http://www.statistik.at/web_de/klassifikationen/regionale_gliederungen/strassen/index.html