Wrangle OpenStreetMap Data

Project 3: Data Analyst Nanodegree

Jorge Santamaría Cruzado

November 22, 2015

1 Data Analysis

1.1 Collecting the data

I've downloaded the pregenerated data file for Madrid at Spain. It can be found at this link [1]. The uncompressed file has a size of 696Mb and it also includes data from Madrid satellite towns like Fuenlabrada, Leganes, Getafe, etc...

1.2 Problems Found

1.2.1 Memory leaks

The first problem was the .osm file. It was too big for my computer to handle, so it keeps eating up all available memory and crashing it.

I needed to write a new function to wrap iterparse called _iter_xml_file 1. This function will clear each node after it's processed to free the memory it was using. Right now, parsing the entire file only uses a few hundreds of Mb.

1.2.2 Street names

The biggest problem so far was the street names. Most of them contained abbreviations. As an example the word "road" is "carretera" in Spanish, and I keep finding abbreviations of it such as "Ctra", "Cr", "Ctra.", etc...

I wrote a mapping for this abbreviations 4 to the correct word. I also added some mapping to manual fix well known street names 5 at this town like "Gran Vía" to "Calle Gran Vía".

At last, I added everything to the function that will handle the street names 6.

1.2.3 Telephones

In Madrid, all the telephone numbers starts with 91 (+34 for the country code) so I wrote a function to check them all, using regular expressions 7. I also wanted uniformity on them so I removed all the whitespaces and add +34 at the beginning on the ones that didn't have it.

I did the same thing for the mobile phones (they all starts with 6 and have 9 digits).

1.2.4 Postal Codes

All postal codes in Madrid starts with 28. However I found a lot of them starting with different numbers. I filter them all out, as they are clearly wrong 8.

1.3 Data overview

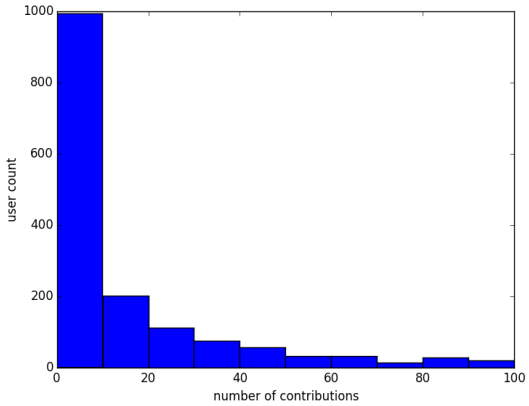
Once I have all the data on my database 2.4, it's much faster and simpler to take a look at it.

```
// number of documents

    > madrid.find().count()
    3664079


// number of nodes

    > madrid.find({'type': 'node'}).count()
    3270804


// number of ways

    > madrid.find({'type': 'way'}).count()
    393275


// number of users

    > madrid.aggregate([{'$group': {'_id': '$created.user'}},
                        {'$group': {'_id': true,
                                    'count': {'$sum': 1}}}])
    { "result" : [ { "_id" : true, "count" : 2092 } ], "ok" : 1 }


// top one user

    > madrid.aggregate([{'$group': {'_id': '$created.user',
                                    'num_contrib': {'$sum': 1}}},
                        {'$sort': {'num_contrib': -1}},
                        {'$limit': 1}])
{
    "result" : [
            {
                    "_id" : "carlosz22",
                    "num_contrib" : 378896
            }
    ],
    "ok" : 1
}

// top ten amenities


   > madrid.aggregate([
            {'$match': {'amenity': {'$exists': true}}},
            {'$group': {'_id': '$amenity', 'count': {'$sum': 1}}},
```

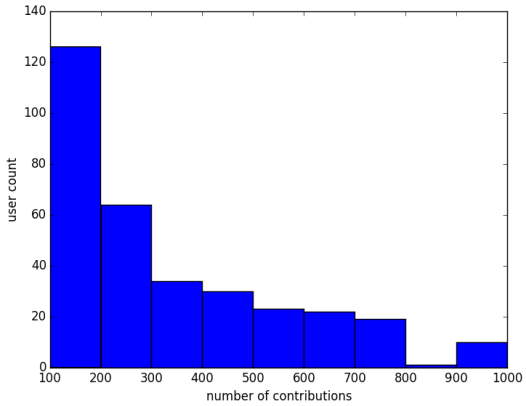
```
        {'$sort': {'count': -1}},
        {'$limit': 10}])
```


```
{
    "result" : [
            {
                    "_id" : "parking",
                    "count" : 4510
            },
            {
                    "_id" : "school",
                    "count" : 3360
            },
            {
                    "_id" : "restaurant",
                    "count" : 3177
            },
            {
                    "_id" : "pharmacy",
                    "count" : 2183
            },
            {
                    "_id" : "bank",
                    "count" : 1564
            },
            {
                    "_id" : "recycling",
                    "count" : 1281
            },
            {
                    "_id" : "bar",
                    "count" : 1190
            },
            {
                    "_id" : "drinking_water",
                    "count" : 1000
            },
            {
                    "_id" : "cafe",
                    "count" : 890
            },
            {
                    "_id" : "place_of_worship",
                    "count" : 858
            }
    ],
    "ok" : 1
}
```

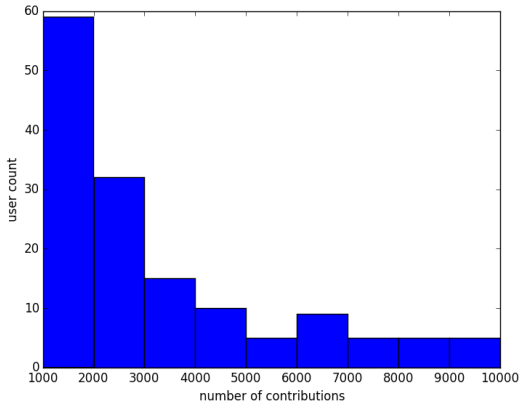
To have a better look at the user contribution stats, I will draw the user contribution histogram using pymongo and matplotlib.pyplot. As expected, the relation between user count and number of contributions seems to be exponential, with many users having a few contributions, and only a few having more than 10 thousand contributions.



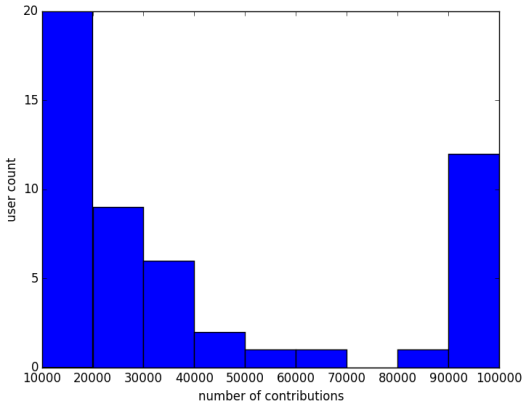
(a) Users with less than 100 contributions



(b) Users with a number of contributions between 100 and 1000

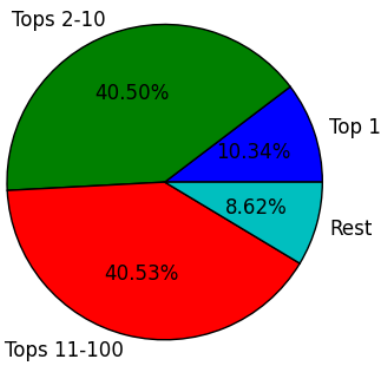


(c) Users with a number of contributions between 1000 and 10k

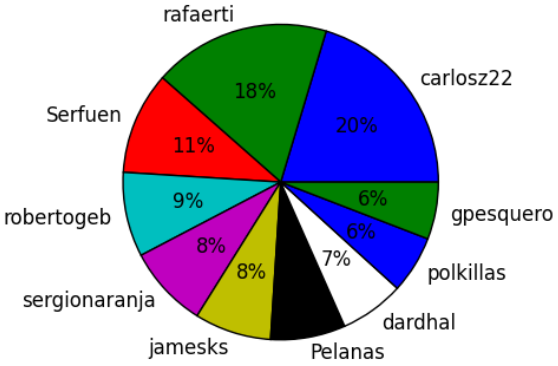


(d) Users with more than 10k contributions. Note: the last bin includes all users with more than 90k contributions

It is clear now that about half of the users have less than ten contributions. Drawing a pie plot of the total contributions shows even more information, looks like about 50% of the contributions are made by only ten users, and only a hundred users make more than 90% of the contributions.



(a) All users

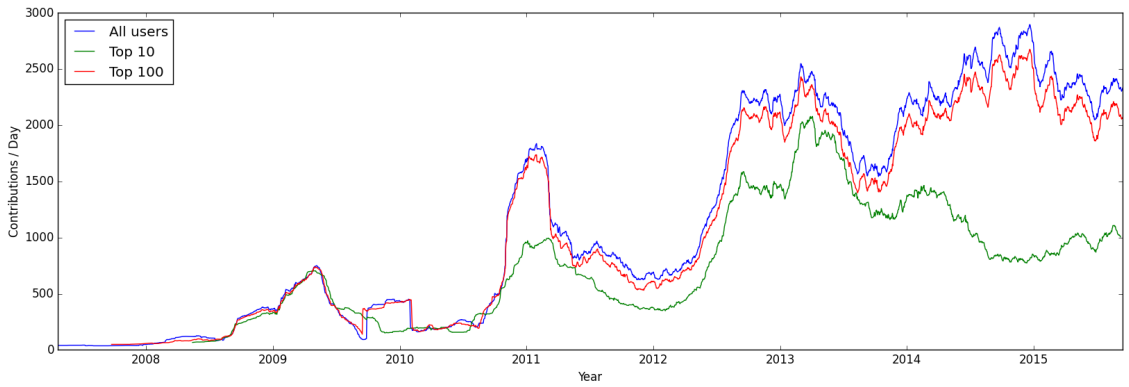


(b) Top ten user contribution

Plotting the contributions per day versus the time, it shows that top 10 users were making most of the contributions until the beginning of 2014 but after that it looks like those users took a step back, and top 100 took their place.

It looks that the graphic have some peaks over time, maybe related to OpenStreetMap adding new features.

Note that I draw the plot using a moving average of 100 days to make the graph smoother. Otherwise it will be much more difficult to get any meaning from it.



1.4 Additional ideas

1.4.1 Bar density

Knowing this town, I was expecting "bar" to be the top amenity at Madrid by far, however, it looks that there are more "parkings" than "bars" here.

If I extend the "bar" definition to "some place where you can consume a drink" I can also include in that "restaurant", and "cafe". Adding the three together I got 5257 coincidences, which looks more like the result I was expecting.

There are 1190 bars in Madrid, as shown in section 1.3, so the proportion of bars in all the drinking establishments is 0.23.

Maybe that's only on my district (university) that the proportion of bars is different from other areas in town.

Let's check if the proportion of bars is different on the university district. It has coordinates (40.435183N, -3.714592E). I use this python script 9 to find out how many of each type are within 1 km of that location. (Direct use of mongo aggregation for this was throwing exceptions due to document max size exceeded on the $geoNear phase)

Once I run it, I see that there are 167 places to have a drink there, and 34 of them are bars, which surprisingly gives me a proportion of 0.20 when I was expecting something higher than the whole town proportion.

Running a fast χ^2 test with my null hypothesis been that there is not significant difference between the proportion of bars at the university district and the proportion of bars in town, I get:

$$\chi^2 = 0.003, \quad p \simeq 1$$

So I can keep the null hypothesis. There is no significant difference on the proportion of bars on my district.

1.4.2 Business recommendations

The $geoWithin operator looks like a very powerful one, it comes to my mind that it wont be hard to get the town population density from somewhere and then use the data on my database to build some kind of business recomendator system.

It would work this way, if someone want to open a business of some kind at this town, it would be easy to check the competitors density in all the areas of the town using that operator and then weight it in some way with the population density of that area to see if that's a good idea opening it there.

1.4.3 Improving the data

Looking at the data it is clear that there are many points missing. Also, even if the node exists at OpenStreetMap, sometimes its data is wrong, maybe because it is outdated or simply bad user input.

The so long discussed gamification feature will surely bring more users who will make a bigger number of contributions and help with the missing points problem, but I care most about quality over quantity here.

To address the quality problem I think it would be great to expand the gamification feature to have some "karma" system. The contributions will only be published if the user has enough "karma" (gained by accurate contributions to the map and upvotes). If the user "karma" is lower than some treshold, more users will be needed to confirm that node before it's published.

Obviously, this will lower the contribution rate (now we need maybe 3 users to add a single node if the have a low "karma") so, in order to keep it, OSM would have to give better incentives to the contributors.

I think is worth to consider displaying some ads at the website, and then split the earnings between the top contributors, that will surely boost the rate of accurate contributions.

Maybe making a premium account for local bussiness (display them on bigger font or something) will also help to improve the contributors incentives, and that way ads can be avoided.

2 Code samples

2.1 Auditing the data

2.1.1 Utils Functions

The default iterparse function was eating up all the memory available and crashing the computer a few seconds after starting, so I need to wrap it with _iter_xml_file to make it clear each element after yielding it.

```
def _iter_xml_file(filename):
    context = iterparse(filename, ('start', 'end'))
    for event, elem in context:
        if elem.tag in ('node', 'way', 'relation'):
            if event == 'end':
                yield event, elem
                elem.clear()

    del context
```

Listing 1: iter_xml_file

2.1.2 Viewing the data

Each element can have multiple tags with key, value pairs. I use this functions to count the number of keys along all the file, so I have a better idea of which of them are important

```
from collections import Counter

def _view_tags(item):
    # Turn the tags k, v pairs into a python dict
    return dict((t.attrib['k'], t.attrib['v'])
                for t in item if t.tag == 'tag')


def view_tags(filename):
    # Count the number of tags each element has
    output = {}
    for event, element in _iter_xml_file(filename):
        if element.tag in elements:
            tags = _view_tags(element)
            if element.tag in output.keys():
                output[element.tag].update(tags.keys())
            else:
                output[element.tag] = Counter(tags.keys())
    return output
```

Listing 2: view_tags

```
def view_tag_values(tag_k):
    # View all the possible values of an specific tag
    return [t.get('v').title()
            for event, element in _iter_xml_file('madrid_spain.osm')
            for t in element
            if t.tag == 'tag'
            if 'k' in t.attrib.keys()
            if t.get('k') == tag_k]
```

Listing 3: view_tag_values

2.2 Cleaning the data

2.2.1 Cleaning street names

```
valid_street_types = {
        'Avenida': ['Avda.', 'Avda', 'Avd', 'Av.', 'Av',],
        'Calle': ['C.', 'C/', 'C/.'],
        'Paseo': [],
        'Camino': [],
        'Rinconada': [],
        'Bulevar': [],
        'Ronda': [],
        'Glorieta': [],
        'Pasaje': [],
        'Corredera': [],
        'Cuesta': [],
        'Carretera': ['Cr','Crta', 'Ctra', 'Ctra,', 'Ctra.', 'Ctra:'],
        'Travesia': ['Travesia',],
        'Autovia': ['Autovia', ],
        'Plaza': ['Pza',],
        'Costanilla': [],
        'Canada': [],
        'Plazuela': [],
        'Autopista': ['Atp'],
        'Carrera': [],
        'Vereda': [],
        }
```

Listing 4: Valid types mapping

```
manual_fix_street_names = {
        'Gran Via': 'Calle Gran Via',
        'Octavio Paz': 'Calle Octavio Paz',
        'Miguel Angel Asturias': 'Calle Miguel Angel Asturias',
        'Antonio Buero Vallejo': 'Calle Antonio Buero Vallejo',
        'Elfo': 'Calle Elfo',
        'Ginebra': 'Calle Ginebra',
        }
```

Listing 5: Manual fix mapping

```
def clean_street(street_name):
    def reverse_search(street_type, valids):
        try:
            return next(key for key, value in valid_street_types.items()
                        if street_type in value)
        except StopIteration:
            return None

    def fix_street_type(street_type, valids):
        if street_type in valids.keys():
            return street_type
        return reverse_search(street_type, valids)

    try:
        street_name = manual_fix_street_names[street_name]
    except KeyError:
        pass


    street = [x.strip() for x in street_name.split(' ')]
    street_type = fix_street_type(street[0], valid_street_types)
    street_name = ' '.join(street[1:])
    return street_type  + ' ' + street_name if street_type else None
```

Listing 6: clean_street_names

2.2.2 Fix telephone numbers

```
cellphone_pattern = re.compile('^6[0-9]{8}$')
phone_pattern = re.compile('^91[0-9]{7}$')

def _fix_telephone(phone, pattern):
    phone = phone.replace(' ', '')
    if phone.startswith('+34'):
        phone = phone[3:]
    if pattern.match(phone):
        return '+34 ' + phone
    return None

def fix_cellphone(cellphone):
    return _fix_telephone(cellphone, cellphone_pattern)

def fix_phone(phone):
    return _fix_telephone(phone, phone_pattern)
```

Listing 7: fix_telephone

2.2.3 Extra regular expressions

```
postcode_pattern = re.compile('^28[0-9]{3}$')
email_pattern = re.compile('^[a-zA-Z0-9_.+-]+@[a-zA-Z0-' +
                           '9-]+\.[a-zA-Z0-9-.]+$')
```

Listing 8: Extra regular expressions

2.3 Extra scripts

```
import pymongo

client = pymongo.MongoClient('localhost', 27017)
col = client.osm.map_data
items = list(col.find({'position':
                           {'$geoWithin':
                               {'$centerSphere':
                                    [[-3.714592, 40.435183],
                                     1/6378]}}}))
bars = len([x for x in items
            if 'amenity' in x.keys()
            if x['amenity'] == 'bar'])

cafes = len([x for x in items
             if 'amenity' in x.keys()
             if x['amenity'] == 'cafe'])

restaurants = len([x for x in items
                   if 'amenity' in x.keys()
                   if x['amenity'] == 'restaurant'])
```

Listing 9: Count bars, restaurants and cafes

2.4 Importing the data into the database

This script will clean all the data using the functions from above and insert it into the database.

```
from clean import *
from datetime import datetime
from audit import _iter_xml_file
from pprint import pprint

import copy
import pymongo

# This script will clean all the data and insert it into the
# database
#
# The document format will be
#
# document = {
#         'created': {
#                 'user': 'Jorge',
#                 'timestamp': datetime.datetime(2015, 11, 20),
#                 'version': 1,
#                 .....
#                 }
#         'address': {
#                 'street': 'Calle Gran Via',
#                 'housenumber': 33,
#                 ....
#                 }
#         'contact': {
#                 'email': 'jsantamariacruzado@gmx.es',
#                 'phone': '+34 911234567',
#                 .....
#                 }
#         'amenity': 'theatre',
#         'name': 'whatever',
#         .....
# }


def _get_item_tags(item):
    # Transform all the tags of the item into a python dictionary
    # so we can handle them better
    return dict((child.get('k').strip(), child.get('v').strip())
                for child in item
                if child.tag == 'tag'
                if child.get('k')
                if child.get('v'))


def _get_node_refs(item):
```

```
    # Put all node references into a list
    return [child.get('ref') for child in item if child.tag == 'nd']

def _shape_address(tags):
    # Will populate the address dictionary with the
    # available items
    address = {}
    for key, value in tags.items():
        if (key.startswith('addr:')) and (key.count(':') == 1):
            if 'street' in key:
                address['street'] = clean_street(value)
                if not address['street']:
                    return None
            elif 'housenumber' in key:
                if housenumber_pattern.match(value):
                    address['housenumber'] = value
            elif 'postcode' in key:
                if postcode_pattern.match(value):
                    address['postcode'] = value
            else:
                address[key.split(':')[-1]] = value
    return address


def _shape_contact(tags):
    # Will populate the contact dictionary with the available
    # items
    # We only care here about the website, phones and email
    contact = {}
    for key, value in tags.items():
        if any(x in key for x in ('cellphone', 'mobile')):
            mobile = fix_cellphone(value)
            if mobile:
                contact['mobile'] = mobile
        elif 'email' in key:
            if email_pattern.match(value):
                contact['email'] = value
        elif 'contact:phone' in key:
            phone = fix_phone(value)
            if phone:
                contact['phone'] = phone
        elif 'website' in key:
            contact['website'] = value

    return contact

def _shape_created(attributes):
    # Will populate the created dictionary with
    # the available items
    keys = ['version', 'changeset', 'timestamp', 'user', 'uid']
    created = {}
```

```
    for k in keys:
        try:
            created[k] = attributes[k]
        except KeyError:
            pass
    return created


def _shape_other(tags):
    # Will get all other items we are not using on
    # address or contact dictionary
    return dict((key, value) for key, value in tags.items()
                if not (key.startswith('addr:'))
                if not (key.startswith('contact:')))


def _get_attributes(item):
    # Get the attributes of the item and translate
    # position and timestamp into python objects if any
    attributes = copy.deepcopy(item.attrib)
    attributes['timestamp'] = datetime.strptime(attributes['timestamp'],
                                                '%Y-%m-%dT%H:%M:%SZ')
    try:
        attributes['position'] = [float(attributes.pop('lon')),
                                  float(attributes.pop('lat'))]
    except KeyError:
        pass
    return attributes


def _shape_item(item, is_way=False):
    # Create the document from the xml item
    tags = _get_item_tags(item)
    attributes = _get_attributes(item)

    other = _shape_other(tags)
    address = _shape_address(tags)
    contact = _shape_contact(tags)
    created = _shape_created(attributes)

    base = dict((key, value)
                for  key, value in attributes.items()
                if key not in created.keys())

    if other:
        base.update(other)
    if address:
        base['address'] = address
    if contact:
        base['contact'] = contact
    if created:
```

```
            base['created'] = created

        if is_way:
            base['refs'] = _get_node_refs(item)
            base['type'] = 'way'
        else:
            base['type'] = 'node'

        return base

    def shape_node(node):
        return _shape_item(node)

    def shape_way(way):
        return _shape_item(way, True)

    def update_database():
        client = pymongo.MongoClient('localhost', 27017)
        collection = client.osm.map_data
        for event, elem in _iter_xml_file('madrid_spain.osm'):
            document = None
            if elem.tag == 'node':
                document = shape_node(elem)
            elif elem.tag == 'way':
                document = shape_way(elem)
            if document:
                collection.insert_one(document)

    if __name__ == '__main__':
        update_database()
```

References

[1] madrid_spain.osm file https://s3.amazonaws.com/metro-extracts.mapzen.com/madrid_spain.osm.bz2