OpenStreetMap Project - Data Wrangling with MongoDB

Egon Kirchof

Map area: Barcelona province, Spain

1. Problems Encountered in the Map

First thing we´ve done after downloading the map was run some code (*code 1*) to find out which tags and attributes there are in it and also to get some sample of the data in “tag” tags (“k” and “v” pairs) so we could audit it (*result* 1).

From our results we have identified some problematic fields that we then tried to clean: street name (addr:street), house number (addr:housenumber), postcode (addr:postcode), city (addr:city), phone, capacity and elevation (ele).

Let´s talk about each one of these fields in the following paragraphs.

Street name (addr:street)

In Barcelona, street names follow the format <street type> <name>.

Some examples we gather from *code 1* are:

"Avinguda d'Antoni Griera", 'Rambla de Catalunya', "carrer Riera de l'Escorxador",

We have seen some ‘street types’ were abbreviated, like ‘c/’ for ‘carrer’ or ‘av.’ for ‘avinguda’ and we have decided to change them to the full form in order to help uniformizing the data in this field.

Also, we have seen in *result 1* some instances of this field where a street number would follow the street name, in the format: <street type> <name>, <number>.

We have decided to remove it from the field, since there is another field with that information (housenumber).

In *code* 3 we have compiled a list of street types by getting the first word at the beginning of the field and saving it in *result 3* and we have used it in *code* 5 in order to change abbreviations to their corresponding long forms:

House number (addr:housenumber)

By inspecting the sample in *result 1* we have seen some values in this field that are not numeric.

We then run *code 2* to see which values in this field are not a number and have found a variety of formats,

Some examples from *result 2* are as follows:

'10 Int', '10, 11', ‘'10-18', ‘sn’, '29/31','Angels Barcelo', ‘1 Esc 12',

We decided to clean this field by keeping only the first number at the beginning of the field and not saving the field when its content has not a number (like ‘sn’).

In *code 5* we clean it with a simple function which returns the number at the beginning of the field.

Postcode (addr:postcode)

After researching the web, we have found out that in Barcelona all postcodes have 5 digits and all start with ‘08’.

In *code 2* we check for any value for this field which doesn´t comply to this rule and the results we got are the followings:

'addr:postcode': ['72', '08037 BCN', '089809', '08','08','08','8007', 'Barcelona']

We decided to change ‘08’ and ‘Barcelona’ to ‘08001’ which is a general postcode for Barcelona.

We could not decide what is the right value for ‘72’ and ‘089809’ so we remove them from the data.

The way we accomplish this cleaning was with a substitution table in *code 5*:

```
problems = {'72':'','08037 BCN':'08037', '089809':'','08':'08001', '8007': '08007', 'Barcelona':'08001'}

if value in problems:
        return problems[value]
else:
        return value
```

City (addr:city)

We have seen a field “addr:city” in the dataset and started to wonder how to know if the content of this field would be correct.

Looking at the internet, we’ve found an web site from the spanish government where a list of all the cities for the province of Barcelona was available and we have used it (municipios barcelona.csv) to check the accuracy of the field city.

In *code 4* we have loaded the csv file containing the list of cities and have parsed our map data checking every instance of ‘addr:city’ to see if the content of this field would be in that list.

The output (*result 4*) from this code shows that only 20 city names are not in the list.

Some of them were simple misspellings (like ‘bacelona’ for Barcelona), the others we decided to remove from the dataset when we clean and save the data as a json file (*code 5*).

Phone

As we inspect values for the field phone we have noticed that a variety of formats have been used to insert this information, like these examples we get from *result 1*:

'+34 93 313 4666','+34 93 396 80 31','+34 93 441 3467',+34 93 479 29 68','+34 931 190 854',

We have decided to make the field uniform by removing the international code (+34) and spaces between digits and getting the first number in the field when more than one was available.

Researching the phone format for Spain we discovered that a valid phone number should start with 9 or 6 and be 9 digits long.

So we run some regular expressions in *code 2* to see if some values in this field don´t follow these specifications.

Some values we can see in *result 2* are:

'+34 93 229 082', '677780096, 935142424, 932212202','933770606 -.934744208',
'933234651 625194474', '+34933182684;+34933182388',
'mobile antenna', 'pole', '+34 9369702511',

The fields with non-numeric values, numbers not starting with 9 or 6 or with less (or more) than 9 digits were not included in the final dataset.

Capacity and elevation (ele)

We decided to clean these fields as an exercise, because all other fields we were dealing with are strings.

We did test capacity and elevation to see if their values are numerics, integer and float respectively.

Some values in capacity are a range in the format <n>-<n> and one value in elevation was a negative number.

For capacity, we clean the field by getting the first number and ignoring the second one, when we find a range.

For elevation, we have ignored negative values.

We accomplish the cleaning with regular expressions in *code 5*: "^\d+" for capacity and "^\d+[.,]?(\d+)?" for elevation.
Since in Spain the decimal separator is “,” the regular expression tests for both “.” and “,” as a decimal separator.

2. Data Overview

Some basic information we gather about the dataset are presented in this section.

We have used **pymongo** for querying the Mongo database.

All the queries we run against the database are in *code 6* and the results we got can be seen in *result 6*.

We have created a function **q(title,pipeline)** for querying the database; in this section we present the function call and the result we got.

The interaction with the database is done with one line of code:

```
result = db.map.aggregate(pipeline)
```

Basic information

File sizes

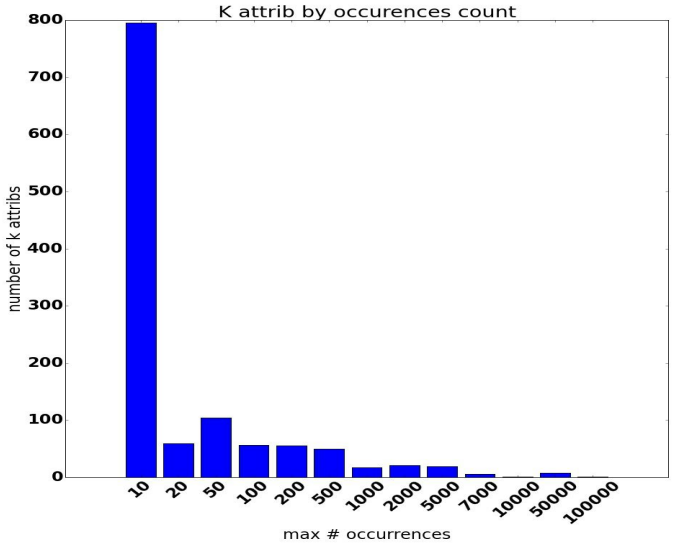
barcelona_spain.osm: 179Mb
barcelona.json: 211Mb
Mongo database size: 283Mb

General numbers from the XML file

As we can see in *result 1*, there are 1192 different kinds of “keys” in “tags”.

The ten most frequent ‘keys’ are: highway (60162), name (41153), building (32609), oneway (22820), addr:street (14209), source (13961), addr:housenumber (12627), addr:city (12192), amenity (11037) and addr:postcode (7061).

In the following graph we can see most “keys” appear less than 10 times in the file.



Number of documents

```
q( "# of documents",        [ {"$group": {"_id": None, "count": {"$sum":1}}}  ] )
```

```
[{u'_id': None, u'count': 959228}]
```

Number of nodes

```
 q( "# of 'nodes'",  [
            { "$match":{"element_type":"node"}} ,
            {"$group": {"_id":"$element_type", "count": {"$sum":1}}}   ] )

[{u'_id': u'node', u'count': 849832}]
```

Number of ways

```
q( "# of 'ways'", [
            { "$match":{"element_type":"way"}},
            {"$group": {"_id":"$element_type", "count": {"$sum":1}}}   ] )

[{u'_id': u'way', u'count': 109396}]
```

Number of unique users

```
q( "# of unique users",  [
            {"$group": {"_id":"$created.user", "count": {"$sum":1}}},
            {"$group": {"_id":None, "count": {"$sum":1}}}          ] )

[{u'_id': None, u'count': 1673}]
```

Top five contributor

```
q( "top five users",[
            {"$group": {"_id":"$created.user", "count": {"$sum":1}}},
            {"$sort": {"count": -1}}, { "$limit": 5}            ] )

[{u'_id': u'josepmunoz', u'count': 95124},
 {u'_id': u'DanielBautista', u'count': 80564},
 {u'_id': u'EliziR', u'count': 61180},
 {u'_id': u'woodpeck_repair', u'count': 33164},
 {u'_id': u'moogido', u'count': 31748}]
```

Top five amenities

```
q( "top five amenities", [
            { "$match": { "amenity": {"$exists":1}}} ,
            {"$group": {"_id":"$amenity", "count": {"$sum":1}}},
            {"$sort": {"count": -1}}, { "$limit": 5}            ] )

[{u'_id': u'parking', u'count': 1396},
 {u'_id': u'restaurant', u'count': 988},
 {u'_id': u'drinking_water', u'count': 754},
```

{u'_id': u'bench', u'count': 680},
{u'_id': u'school', u'count': 592}]

Some more queries

Number of amenities by city and type

```
q( "# amenities per city", [
        { "$match": {"amenity":{"$exists":1},"address.city":{"$exists":1}}} ,
        {"$group": {"_id": {"acity":"$address.city","amenity":"$amenity"}, "count": {"$sum":1}}},
        {"$sort": {  "_id.amenity": 1, "_id.acity":1  }}            ] )
```

The result from this query is too long so we only present a few lines here.
The complete result can be found in *result 6*.

[{u'_id': {u'acity': u'Barcelona', u'amenity': u'animal_shelter'}, u'count': 1},
{u'_id': {u'acity': u'Barcelona', u'amenity': u'arts_centre'}, u'count': 2},
{u'_id': {u'acity': u'El Prat de Llobregat', u'amenity': u'arts_centre'}, u'count': 1},
{u'_id': {u'acity': u'Sant Cugat del Vall\xe8s', u'amenity': u'arts_centre'}, u'count': 1},

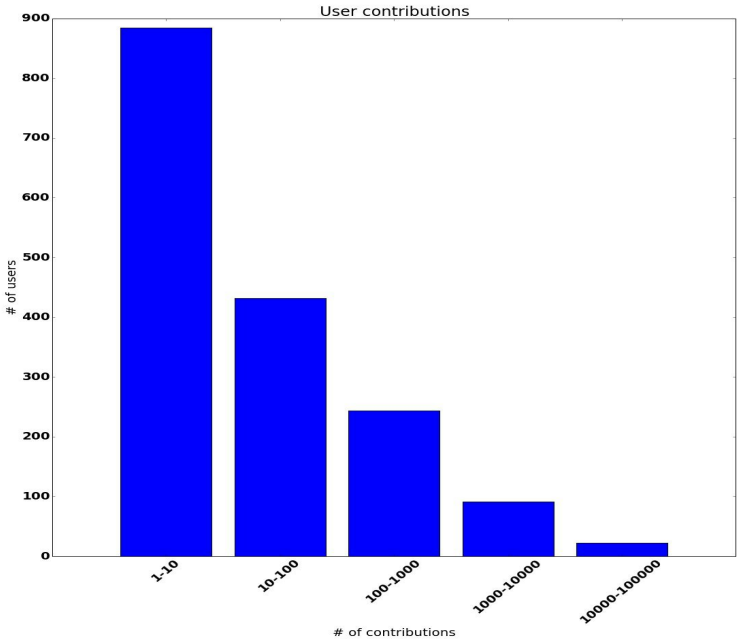
Contributions per user

```
q( "#contributions per users", [
        {"$group": {"_id":"$created.uid", "count": {"$sum":1}}},
        {"$sort": {"count": 1}}       ] )
```

The result for this query is very long so we only present a few lines.
The complete result can be found in *result 6*.

[{u'_id': {u'acity': u'Barcelona', u'amenity': u'animal_shelter'}, u'count': 1},
{u'_id': {u'acity': u'Barcelona', u'amenity': u'arts_centre'}, u'count': 2},
{u'_id': {u'acity': u'El Prat de Llobregat', u'amenity': u'arts_centre'}, u'count': 1},
{u'_id': {u'acity': u'Sant Cugat del Vall\xe8s', u'amenity': u'arts_centre'}, u'count': 1},
{u'_id': {u'acity': u'Badalona', u'amenity': u'bank'}, u'count': 19}, {u'_id': {u'acity': u'Barcelona', u'amenity': u'bank'}, u'count': 20},
{u'_id': {u'acity': u'Bellaterra', u'amenity': u'bank'}, u'count': 1},
{u'_id': {u'acity': u'Cornell\xe0 de Llobregat', u'amenity': u'bank'}, u'count': 1},
{u'_id': {u'acity': u'Sabadell', u'amenity': u'bank'}, u'count': 1},

Using the result of this query we have created a bar graph showing the number of contributions users have done.



From the graph we can see most users have done between 1 and 10 contributions.

3. Additional Ideas

One way we could increase the quality of our data would be validating some fields found in the map by comparing it with information available from other sources.

For instance, we could check the validity of entries related to amenities by checking the phone provided in the map database to see if we can find the same information in the database of a website like yelp.

Yelp allows us to do this kind of search through an easy to use api which returns a Json document with fields like **location:adress**, which we could compare to **adress:street** in our database .

We could create a function in python that queries the website according to a list of documents containing phone numbers and then try to compare their locations to see if there is a match.

One problem we would have is if the document in our database doe not have a field **adress:street** and another situation that could happen is we don´t find a record for that phone in yelp.

Both situations happened when we have queried our database for two instances of documents which have a field phone:

> db.map.aggregate([{$match:{phone:{$exists:true}}},{$limit:2}])

{ "_id" : ObjectId("55ecbf031c65cd72b5c617b1"), "website" : "http://www.campinge
lvedado.com/", "fax" : "+34 935729621", (…)

"phone" : "935729026", "element_type" : "node", "tourism" : "camp_site", "id" : "26861131" }

{, "phone" : "936720020", "element_type" : "node", "source" : "mityc 20100729", "address" : { "street" : "Avinguda Catalunya", "housenumber" : 135 }, "operator" : "Repsol", "id" : "287006756
", "is_in:municipality" : "la Palma de Cervelló", "fuel:diesel_c" : "yes" }

The first document did not have a field **adress:steet** and also was not found in yelp when we queried the website (the information returned was truncated to save some space):

http://api.yelp.com/v2/phone_search/?phone=935729026&cc=ES

{ "region": null, "total": 0, "businesses": [] }

http://api.yelp.com/v2/phone_search/?phone=936720020 &cc=ES

{ "region": null, "total": 1, "businesses": [{ (...) "phone": "+34936720020",
"location": { (...)
"address": ["Avenida Catalunya, 135"],
"coordinate": { "latitude": 41.4138603, "longitude": 1.9633501 }, }]
}

As we can see, for the first document we got from our database we can not confirm the validity of the information by this method but for the second one we did have a match.

That means we would still have cases where the validation would not work.

If the address returned by the second query was not the one we have in our database we would have to choose what to do with our document: remove it from the database or simply flag it.

There is no way we can tell which database have the most accurate information.

Another problem we could face is the case the business phone is updated in the yelp website but not in the map or vice versa.