OpenStreetMap Project: Data Wrangling with MongoDB

Sergey Mushinskiy (cepera.ang@gmail.com)

Map area: Tyumen, Russia.

I choose Tyumen city from Russia because I wanted to use some Russian city but most of them to small and only Tyumen is big enough but not too big as Moscow for example.

Problems encountered in your map

Street names

After downloading file, I ran some audit code against it. Here, in Russia, we almost always name streets using street type (such as “street”, “road” etc) at the first place. But in some cases, when street name is adjective, we place street type at the end. So, our audit script can’t be easily tweaked to account all this differences and I just dumped all the names and checked it with my eyes. Seems like there is no such problems with street names as in Lesson 6. Almost all streets named correctly. Few suspicious streets may be just local way to name streets (example: “4 км Червишевского тракта” which means “4’s km Chervishevskogo trakta”, this is strange name but it is a real official name of the street).

Postal codes

Another thing to check is postcodes. All postcodes it Tyumen have to be 6-digit numbers starting with 6xxxxx (exact range can be found here http://info.russianpost.ru/servlet/department), so we can check if there are invalid numbers or even other strings and as we don’t know actual postcode so just clear this field:

```
def update_postcode(old_postcode):
    """

    :type old_postcode: str
    """
    new_postcode = old_postcode
    try:
        if int(old_postcode) < 625000 or int(old_postcode) > 997000:
            new_postcode = ""
    except:
        new_postcode = ""
    return new_postcode
#…  skipped
elif is_postcode(tag):
    temp_postcode = update_postcode(tag.attrib['v'])
    if temp_postcode != tag.attrib['v']:
        invalid_postcodes.append(tag.attrib['v'])
pprint.pprint(invalid_postcodes)
```

We can see following output:

['<различные>', '38', '36', '40', '5', '7', '6', '8', '12', '<различные>', '<различные>', '90']

Phone numbers

Many amenities in Tyumen dataset have their phone numbers set. However, all this numbers have different formatting. Let’s fix it. We want to have all phone numbers in common format: for local city numbers it should be +7(3452)12-34-56, for all other numbers: +7(123)456-78-90. Following code allows us to do it in a second:

```
def update_phone(old_phone):
    # this script is quick and dirty as it didn't check some corner cases and
```

```
    # only work fine for particular dataset, so don't expect much from it

    # remove all non-numeric symbols except +
    new_phone = re.sub("[^0-9\+]", "", old_phone)

    # for local city phones we want to have format as such: +7(3452)12-34-56
    if new_phone.find("3452") != -1:  # ! here is a catch -- what if phone has 3452 not only in area code
part?
        new_phone = new_phone.replace("3452", "(3452)")

        # check if we have leading +7 or not
        if new_phone.find("(3452)") == 0:
            new_phone = "+7" + new_phone

        # +7(3452)123456 -- number should be that way at this moment
        # +7(3452)12-34-56 -- final phone number
        new_phone = new_phone[0:10] + "-" + new_phone[10:12] + "-" + new_phone[12:14]
        return new_phone
    if new_phone.find("+7") == -1 and new_phone.find(
            "7") == 0:  # another catch here -- what if we have just number starting from 7?
        new_phone = "+" + new_phone

    # other phones go in different format
    # so we have to change it from +7xxx1234567 to +7(xxx)123-45-67
    new_phone = new_phone[0:2] + "(" + new_phone[2:5] + ")" + new_phone[5:8] + "-" + new_phone[8:10] + "-"
+ new_phone[10:12]
    return new_phone
```

Websites

Another common contact information is website. This is just an esthetic tweak but I love my data to be as tidy as possible. So I want to append “http://” for any website where is no such prefix. Easy:

```
def update_website(old_website):
    new_website = old_website
    if old_website.find("http") == -1:
        new_website = 'http://' + old_website
    return new_website
```

Overview of the Data

In this section, we explore dataset a little bit more.

File size

```
tyumen_russia.osm . . . . . 66,329KB

tyumen_russia.osm.json. . . 76,184KB
```

Statistics about data

```
# Number of documents

        > db.osm.find().count()
        351824

# Number of nodes

        > db.osm.find({"type":"node"}).count()
        309762

# Number of ways

        > db.osm.find({"type":"way"}).count()
        41983

# Number of unique users

        > db.osm.distinct("created.user").length
```

```
# Top 10 contributing user

> db.osm.aggregate([{ "$group" : { "_id" : "$created.user", "count" :{ "$sum"
: 1 }}}, {"$sort" : {"count" : -1}}])

{ "_id" : "Oksion", "count" : 138694 }
{ "_id" : "Dennis Apter", "count" : 88443 }
{ "_id" : "RSergei", "count" : 36309 }
{ "_id" : "mavl", "count" : 15433 }
{ "_id" : "pinhead75", "count" : 9991 }
{ "_id" : "cvuneeez", "count" : 7098 }
{ "_id" : "Павел Яринюк", "count" : 6826 }
{ "_id" : "stopa85", "count" : 6277 }
{ "_id" : "alexfrol", "count" : 5296 }
{ "_id" : "savsav", "count" : 2957 }

# Top 20 amenities by type


> db.osm.aggregate({"$match":{"amenity":{"$exists":1}}},
{"$group":{"_id":"$amenity", "count":{"$sum":1}}}, {"$sort" : {"count" : -1}})

{ "_id" : "parking", "count" : 332 }
{ "_id" : "kindergarten", "count" : 104 }
{ "_id" : "fuel", "count" : 96 }
{ "_id" : "school", "count" : 75 }
{ "_id" : "bank", "count" : 51 }
{ "_id" : "cafe", "count" : 45 }
{ "_id" : "hospital", "count" : 44 }
{ "_id" : "public_building", "count" : 41 }
{ "_id" : "restaurant", "count" : 36 }
{ "_id" : "pharmacy", "count" : 34 }
{ "_id" : "drinking_water", "count" : 26 }
{ "_id" : "car_wash", "count" : 25 }
{ "_id" : "bench", "count" : 21 }
{ "_id" : "atm", "count" : 21 }
{ "_id" : "fast_food", "count" : 20 }
{ "_id" : "university", "count" : 20 }
{ "_id" : "place_of_worship", "count" : 19 }
{ "_id" : "fountain", "count" : 18 }
{ "_id" : "post_office", "count" : 17 }
{ "_id" : "marketplace", "count" : 16 }
```

As we can see our data definitely incomplete. For example, number of ATMs for a city with more than six hundred people has to be much more than 21. Same for benches and other amenities. For schools and kindergarten degree of completeness is much more (actually in city about 150 kindergartens and 100 schools). Maybe it comes from bigger public value of them and contributors more likely map school than mere shop or pizza.

Other ideas about the datasets

If we look more carefully throw dataset we can see more room for improvement.

Counting buildings

We can extract additional data from our dataset. In this example we will count all buildings with assigned house number, meaning that this building officially recognized by city authorities. We can use this information for comparison with official sources (KLADR – address registry in Russia) to see how many buildings left to be correctly mapped or assigned correct address in dataset:

```
> db.osm.aggregate({"$match":{"address.housenumber":{"$exists" : 1}}},
{"$group":{"_id":"", "count":{"$sum":1}}}, {"$sor
t" : {"count" : -1}})
{ "_id" : "", "count" : 9665 }
```

If we query total number of buildings in city, there will be much more. What does it means? Is there many buildings in the city that don't have any address or some buildings' address just do not mapped yet? We can't say rely only on OSM dataset – we need some additional sources to check any of this.

```
> db.osm.aggregate({"$match":{"building": "yes"}}, {"$group":{"_id":"is_building", "count":{"$sum":1}}},
{"$sort" : {"count" : -1}})
{ "_id" : " is_building ", "count" : 16902 }
```

Additional address field cleaning and fixing

Additional data cleaning is possible here, but we have to be extremely accurate (and such automated edits are not very welcomed in openstreetmap community). For example, we can see that many addresses are not complete. They may include some part but not another. Above we have 9665 building with house number set but we have only 4509 buildings where country set and just 5341 where city is set (queries below). We can safely assume that any building with some part of address set can be automatically marked as belonging to RU country and Tyumen city.

```
> db.osm.aggregate({"$match":{"building": "yes"}},
{"$group":{"_id":"$address.country", "count":{"$sum":1}}}, {"$sort" :
 {"count" : -1}})
{ "_id" : null, "count" : 12393 }
{ "_id" : "RU", "count" : 4509 }

> db.osm.aggregate({"$match":{"building": "yes"}}, {"$group":{"_id":"$address.city",
"count":{"$sum":1}}}, {"$sort" : {"
count" : -1}})
{ "_id" : null, "count" : 11538 }
{ "_id" : "Тюмень", "count" : 5341 }
{ "_id" : "Луговое", "count" : 8 }
{ "_id" : "Московский", "count" : 4 }
{ "_id" : "Боровский", "count" : 4 }
{ "_id" : "Дербыши", "count" : 2 }
{ "_id" : "Ожогино", "count" : 2 }
{ "_id" : "Дударево", "count" : 2 }
{ "_id" : "Падерино", "count" : 1 }
```

Another thing we can do again about postcodes. In this dataset, we have only about 600 buildings with postcodes set. What we can do? We can check other parts of address and if they are present, we can try to use external databases to find out postcode and add it to map. There are several online sites doing exactly this.

Auditing city infrastructure

Another thing we can do with our dataset is to count other city infrastructure, such as highways and its length (we have all information in node refs but it is more difficult and probably outside the scope of basic analysis. We have to get all nodes for ways representing roads, then find all the node refs for each road and then measure distances between this nodes.) In addition, we can easily count various objects like traffic signals

```
> db.osm.aggregate({"$match":{"highway": "traffic_signals"}},
{"$group":{"_id":"$highway", "count":{"$sum":1}}}, {"$sort" : {"count" : -1}})
{ "_id" : "traffic_signals", "count" : 300 }
```

Or bus stops

```
> db.osm.aggregate({"$match":{"highway": "bus_stop"}}, {"$group":{"_id":"$highway",
"count":{"$sum":1}}}, {"$sort" : {"count" : -1}})
{ "_id" : "bus_stop", "count" : 354 }
```

We can calculate area of different types of land use (for example area of lawns or garages), but it requires some advanced programming and not implemented here. Little surprisingly, we can see even farms in this city (and there is nearly two hundreds of it!)

```
> db.osm.aggregate({"$match":{"landuse": {"$exists":1}}},{"$group":{"_id":"$landuse",
"count":{"$sum":1}}}, {"$sort" : {"count" : -1}})
{ "_id" : "grass", "count" : 912 }
{ "_id" : "residential", "count" : 853 }
{ "_id" : "garages", "count" : 131 }
{ "_id" : "industrial", "count" : 123 }
{ "_id" : "farmland", "count" : 107 }
{ "_id" : "farm", "count" : 81 }
```

Conclusion

This review of OpenStreetMap data shows clearly that typical human-created dataset can be messy and incomplete. Some of the problems can be fixed quite easily (such as website or phone number) and other requires human attention (like correct addressing or adding amenities to map). However, distribution of user contributions shows that just five people can provide 80% of map data for a half-a-million big city (we can see it from queries in overview section -- # of total entries is 351824 and Top 5 contributors ("Oksion" : 138694, "Dennis Apter" : 88443, "RSergei" : 36309, "mavl" : 15433, "pinhead75" : 9991) made 288870 or 82.1% of this entries).

In general, many of problems encountered in this dataset and solutions for them have a long history of discussions in OSM community (here is discussion about correct naming of streets and here is why we have almost all street names correctly specified – it was all Russia conversion project) and even tools developed to help mapping (tool that shows all buildings without addresses).