OpenStreetMap Sample Project

Data Wrangling with MongoDB

Iuşan Andrei

Map Area: Bucharest, Romania

1. Problems encountered in the dataset

After auditing the data, I observed two problems that I will discuss further:

- Miss-spelled street names (that I corrected before inserting the data in MongoDB)
- Incorrectly marked amenities (that appeared while querying the database and I updated them)

The process is discussed in the following sections.

1.1 Auditing

I wrote a code to audit the data. The firs part of the code just outputs information so that I can inspect it. After printing the tags for ways, I realize that the most common tags are:

```
'building': 45929
'highway': 41665
```

This suggests that about half of the `'way'` elements in the dataset represent buildings and about half of them represent roads. I also noticed that the addr:street tag appears 1703 times. I suspect that a building has an address field, but the '`addr:street`' field is not required for a street. The street name appear in the `'name'` tag of the elements that also contain a `'highway'` tag.

I also printed the tags for nodes, and the most common tags are:

```
'source': 107514
'addr:housenumber': 72684
'addr:street': 72629
```

In Romania, the street type appears before the street name (e.g. “Bulevardul Republicii”, while in English this would be “Republicii Boulevard”). I checked for the first word in street names, both in the address field of the buildings and in the name field of the elements with a 'highway' tag. For this dataset, street names are not abbreviated, but a number of misspellings are present. I selected the misspelled words and used the list in the cleaning step.

1.2 Cleaning

Using the output of the audit script, I corrected the street names and saved the file as json. I then uploaded the json file into Mongo db using mongoimport:

```
$ mongoimport -d osm -c bucharest --file bucharest_romania.osm.json
```

The document schema contains the fields `'created'`, `'id'`, `'type'`, and all the tags in the osm object. If `'type'` equals `'way'` we also have `'node_refs'` as an array containing the ids of the nodes that the way references; if type is `'relation'` we have `'members'` as an array of documents that contain references to the members of the relation (a relation references nodes and/or ways). If the element contains an address, I expanded it in a document like this: `'address': {'street': '...', 'housenumber': '...' ...}`. Note that the elements in the address field vary, some elements contain more information than others.

1.3 Querying the database

After inserting the data in MongoDB, I queried the database for shops and found those results.

```
db.bucharest.find({'amenity':'shop'}).count()
6
db.bucharest.find({'shop':{'$exists':1}}).count()
2334
```

This is surprising since I believe the shops should also have the tag 'amenity'. This reveals another problem, but as the data was already in the database at this point, rather than reviewing the previous stages, and clean the data from the xml file while converting it into json, I came up with a method to update all amenities in the database. Here are a few results regarding the number of amenity fields created by the algorithm:

```
'shop': 2334,
'atm': 430,
'parking': 281
```

The way this algorithm works is described in the comments of the source code.

2. Data Overview

This section contains statistics about the dataset, obtained using MongoDB queries.

Size of file:

```
bucharest_romania.osm .......... 206 MB
bucharest_romania.osm.json ...... 227 MB
```

Number of documents:

```
> db.bucharest.find().count()
928429
```

Number of nodes
```
> db.bucharest.find({'type':'node'}).count()
829812
```

Number of ways
```
> db.bucharest.find({'type':'way'}).count()
97165
```

Number of relations
```
> db.bucharest.find({'type':'relation'}).count()
1452
```

Number of users
```
> users = db.bucharest.aggregate([{'$group':{'_id':'$created.user','count':{'$sum':1}}},
{'$sort':{'count':-1}}])
> users = users['result']
> len(users)
749
```

Top 5 contributing users:
```
> users[:5]
[{u'_id': u'razor74', u'count': 272851},
 {u'_id': u'dincaionclaudiu', u'count': 138865},
 {u'_id': u'baditaflorin', u'count': 84928},
 {u'_id': u'Mircea Toader', u'count': 74937},
 {u'_id': u'Romania CLC import bot', u'count': 66724}]
```

Number of shops (after cleaning)
```
> db.bucharest.find({'amenity':'shop'}).count()
2336
```

3. Other Ideas

Another great way of cleaning and standardizing the dataset is to bring in more data from different sources. Since every node is required to have the position and a large number of nodes do not contain address information, it is worth exploring the use of reverse geocoding services, like the Google Geocoding API: https://developers.google.com/maps/documentation/geocoding/

This service allows converting latitude, longitude pairs to addresses. This can be used for adding address information to nodes that do not contain an address, or that contain incomplete information. The service could also be used to automate the process, such that when a new node is added to the map, a script will check it and update the address.

The challenge in this case is to verify the validity of the data. If such a service is used to update a large amount of data, we need to make sure that no additional errors are introduced (like obtaining wrong address for a given point on the map). The benefit of this method is that we obtain already standardized data, and we would therefore introduce clean data into the dataset.

There are also datasets with postal codes available like:

- http://date.gov.ro/organization/posta-romana
- http://www.coduripostale.ro/en

Those datasets can be used to automatically update a large amount of data. The first example is an xml file that can be parsed. In the second example, the data is presented on a website. In order to use this data, a scraper program has to be built. The biggest challenge here involves geocoding. Those datasets map addresses to postal codes, but if the address is missing, we are unable to map the postal code to the node in our osm dataset. The advantage would be that the those datasets are complete.

Conclusion

After inspecting this dataset it is clear that the data is not standardized. The data has been cleaned to some extent in this assignment, but further cleaning is possible. Most elements have been cleaned programatically, but others - those that have unique tags - require special attention if we are to preserve as much of the data as possible.

It should also be noted that the data is incomplete. I presented two ways to significantly improve the dataset:

- Reverse Geocoding
- Using external datasets

If those methods are applied properly together, it should be possible to clean and standardize, and also add a large amount of data.