Data Wrangling Project

OpenStreetMap Project
Data Wrangling with MongoDB

Tommy Siu

Map Area: Hong Kong, China

http://overpass-api.de/query_form.html
Query: (node(22.1505,113.8225,22.5151,114.4102);<;);out meta;

1. Problems Encountered in the Map

I noticed two main problems with the data, which I will discuss in the following order:

- Abbreviated street names
- Inconsistent addresses

Abbreviated Street Names

Firstly I run maputils.audit2() to audit against the data, in order to find out all unexpected street types and the corresponding counts. By printing out the counters, the following abbreviations and strange street types could be identified:

'Rd':8
'St':1
'St.':1
'Circuit\\':1
u'Street\u200e':1

The number is small and the fix is straight forward. The last two are likely typos so I fix them altogether. I also noticed that the street types 'Central', 'East', 'South', 'West' and 'North' have relatively large counts:

'Central':7
'East':15
'South':9
'West':16
'North':6

This naming convention is common in Hong Kong for long streets. A further look into the full street names of each of the above street types showed that there is no abbreviations inside. For example:

```
> types, counts = maputils.audit2('hongkong.osm')

> types['Central']
   {"50-52 Queen's Road Central",
   'Connaught Road Central',
   'Des Voeux Road Central',
   'Healthy Street Central',
   "Queen's Road Central",
   u'\u5e72\u8afe\u9053\u4e2d Connaught Road Central',
   u"\u7687\u540e\u5927\u9053\u4e2d Queen's Road Central"}
```

So I decided to update only the street abbreviations and typos found earlier, by calling the maputils.convert_map('hongkong.osm').

Note that the original data.py stores the OSM element type (i.e. <node> or <way>) as a JSON property named 'type'. However, it ran into problem immediately because some OSM elements already have the key:type tags and the corresponding values will override the 'node'/'way' values I set to the JSON documents. Thus I modified data.py to store the element type as JSON property 'doc_type' instead.

After that, I import the JSON file into a local MongoDB database using dbutils.insert_data('hongkong.osm.json', 'hongkong'), where 'hongkong' is the collection name.

Inconsistent Addresses

In the map data, if a node/way contains an address field, normally it contains a street name too.

```
# Number of nodes/ways with an address and a street name
   > db.hongkong.find({'address.street':{'$exists':1}}).count()
   7713

# Number of nodes/ways with an address but no street name
   > db.hongkong.find({'address':{'$exists':1},
   'address.street':{'$exists':0}}).count()
   1025
```

It is a very common use case to search the map data with a street name. Thus I wondered what those addresses without street names contain. I tried to print out and examine the pattern of address sub-documents from the previous query.

```
# Print all address sub-documents
   > result =
   db.hongkong.find({'address':{'$exists':1},'address.street':{'$exists':0}})
   > for n in result:
```

```
print n['address']
```

Quickly a clear pattern can be seen. The 1025 documents could be roughly separated into two main types:

a) contain addr:housenumber only, e.g. {u'housenumber': u'9-11'}

b) contain addr:housenumber and addr:place, e.g. {u'place': u'Tin Liu New Village', u'housenumber': u'32'}

According to OSM Wiki, we should not only add tag addr:housenumber but without tag addr:street. In case addr:street is not applicable, we could use addr:place. The following query shows the number of addresses with addr:housenumber but no addr:street / addr:place.

```
# Number of nodes/ways with an address but no street or place
> db.hongkong.find({'address.housenumber':{'$exists':1},
'address.street':{'$exists':0},
'address.place':{ '$exists':0}}).count()
783
```

In this project I decided to patch some node documents of the data set. There are much fewer documents if I only consider nodes:

```
# Number of nodes with an address but no street or place
> db.hongkong.find({
'doc_type':'node',
'address.housenumber':{'$exists':1},
'address.street':{'$exists':0},
'address.place':{ '$exists':0}}).count()
55
```

I noticed that many of these nodes have an addr:housenumber starting with H or M, e.g. H22, H34, M1…etc. I further check the changeset of these documents, using a python function in my dbutils.py with regular expression '^[H|M].*' to match only H* or M* housenumbers:

```
> for n in dbutils.find_nodes_by_housenumber(db.hongkong, '^[H|M].*'):
        print 'housenumber=%s, changeset=%s' %
        (n['address']['housenumber'], n['created']['changeset'])
```

All of the above nodes were created by a changeset 7868019, which added houses along the street "Tso Wo Road".

Finally I patch the above nodes in my MongoDB database by adding a proper street name:

> dbutils.update_nodes_street(db.hongkong, 'Tso Wo Road', '^[H|M].*')

Other nodes could be patched in a similar way. I wrote two functions update_nodes_street() and update_nodes_place in dbutils.py, so I can update either the address.street or address.place to patch the addresses.

2. Data Overview

This section contains basic statistics about the dataset and the MongoDB queries used to gather them.

File sizes

hongkong.osm..... 187 MB
hongkong.osm.json .... 268 MB

Number of documents

```
> db.hongkong.find().count()
970126
```

Number of nodes

```
> db.hongkong.find({'doc_type':'node'}).count()
877075
```

Number of ways

```
> db.hongkong.find({'doc_type':'way'}).count()
93051
```

Number of unique users

```
> len(db.hongkong.find().distinct('created.user'))
704
```

Top 10 contributing user

```
> list(db.hongkong.aggregate([
{'$group':{'_id': '$created.user', 'count':{ '$sum':1}}},
{'$sort':{'count':-1}}, {'$limit':10}]))

 [{u'_id': u'hlaw', u'count': 510158},
  {u'_id': u'KX675', u'count': 77302},
  {u'_id': u'bTonyB', u'count': 42562},
  {u'_id': u'jc86035', u'count': 38160},
  {u'_id': u'R17466', u'count': 31927},
  {u'_id': u'Mapier', u'count': 22552},
```

```
 {u'_id': u'moloi', u'count': 20523},
 {u'_id': u'OmniBusSun', u'count': 18989},
 {u'_id': u'ystsoi', u'count': 17322},
 {u'_id': u'minghong', u'count': 15381}]
```

```
# Number of elements created within the last 365 days

> start = datetime.strftime(datetime.today() - timedelta(days=365), '%Y-
%m-%dT%H:%M:%SZ')

> start
'2014-09-13T21:37:13Z'

> db.hongkong.find({'created.timestamp':{'$gte':start}}).count()
183275

# Top 10 element versions

> result = db.hongkong.aggregate([
 {'$group':{'_id':'$created.version','count':{'$sum':1}}},
 {'$sort':{'count':-1}}, {'$limit':10}])

> list(result)
 [{u'_id': u'1', u'count': 695576},
 {u'_id': u'2', u'count': 158320},
 {u'_id': u'3', u'count': 65392},
 {u'_id': u'4', u'count': 23188},
 {u'_id': u'5', u'count': 10305},
 {u'_id': u'6', u'count': 5508},
 {u'_id': u'7', u'count': 3277},
 {u'_id': u'8', u'count': 2385},
 {u'_id': u'9', u'count': 1994},
 {u'_id': u'10', u'count': 860}]
```

3. Additional Ideas

Data Validation

From the dataset, we know that there are 8256 elements (either node or way) which have addr:housenumber. However, 8.8% (728) of those elements have neither addr:street nor addr:place. I guess one of the reasons is the free editing characteristics of OSM, which allows the addition of data regardless whether the structure is adequate or not semantically.

There are already a number of tools to validate OSM data. However, it seems most of the validations are working in user side and the OSM system has no governance at all. While it may be an important idea of OSM (works

like Wikipedia and all users maintains the correctness of data altogether), in my opinion it also seriously affects the usefulness of the data.

I think one possible improvement is to implement a validation pipeline in system side (rather than user side). The validation pipeline could be invoked during data update and any issues associated with the committed data is saved somewhere so it is open and searchable by the public. The validation pipeline itself can be another OSM sub-project which allows public user collaboration and contribution, according to some guidelines and API supported by OSM.

However, there could be some possible issues of this proposal.

Firstly, it is obvious that the data validation process would add a certain amount of workload over the existing OSM servers. The cost depends on the time we like to trigger the validation. Will it be triggered immediately during a changeset upload? Or it will be more appropriate to schedule batch jobs? If we want to have daily scheduled tasks to perform the validation, another decision point is whether we want to do it in region level so data in different regions will be validated in different timeslots to minimize user impact. More importantly, how much data do we need to validate daily? Probably a detailed traffic analysis must be done first in order to plan the extra capacity needed. The rule of thumb is that we need to balance the data accuracy and the user experience as well.

Secondly, the design and implementation of a server side validation system could be very complex if it involves public user contribution, and we allow each of them to contribute part of data validation logic. I think it makes sense to design a mechanism for users to contribute the validation code, rather than implementing it solely by OSM developers, because the users should be the one who are actually familiar with their regions; otherwise it defeats the purpose of OSM being open and owned by users. One main difficulty of this idea is, we need to define some rules for users to follow, like the OSM data editing. However, in this case it is program code rather than data. How can we ensure that one validation program submitted by user A does not contradict with another validation program by user B? What is the appropriate language? Python? Java? Or a custom script-like language? How about different versioning of validation code? If a user submits a newer version of validation program, does the system need to run it again over all previously valid data? I just try to point out a few considerations here. There must be more to come if we really implement such system.

Thirdly, if we have a successful implementation of this system side validation pipeline, it may generate a large amount of data. The validation data itself is something we could further analyze. For example, what is the invalid data rate for particular regions, and the relevant reasons? Certainly we want to understand the implication of the validation data, otherwise we may lose important information and the chance to further improve the system.

Conclusion

After this review of the data I see that a small portion of the node street names contain abbreviation. However, it is hard to draw a conclusion at this stage whether the Hong Kong data is well mapped or not. It is necessary to perform further validations by some publicly available tools, or implement a system-wide validation mechanism in order to ensure the overall mapping quality.