OpenStreetMap Sample Project Data Wrangling with MongoDB

Guo Yading

1. Problems Encountered in the Map

1.1 Running "audit.py"

I downloaded the map of Beijing in China in xml format. After a little digging, I run my audit.py against the xml file and print the result in an organized way.

In [2]:

```
execfile( "audit.py")
```

```
************************************************************
Problem 1: inconsistency in keys
First 2 unicode keys in the "tag" element are:
[u'\u758f\u6563\u4eba\u6570', u'\u9ec4\u5357\u82d1\u5c0f\u533
a']
------------------------------------------------------------
First 2 strings containing hyphen are:
['name:bat-smg', 'name:roa-rup']
------------------------------------------------------------
First 2 strings containing uppercase letters or abbreviated wo
rds are:
['No.', 'currency:CNY']
************************************************************


************************************************************
Problem 2: colon(s) in keys
First 2 strings containing single colon:
['ref:en', 'name:uz']
------------------------------------------------------------
First 2 strings containing duoble colons:
['service:bicycle:retail', 'service:bicycle:repair']
************************************************************


************************************************************
Problem 3: pinyin as names
One tag element containing name in pinyin:
('name', 'Guxiang 20')
************************************************************
```
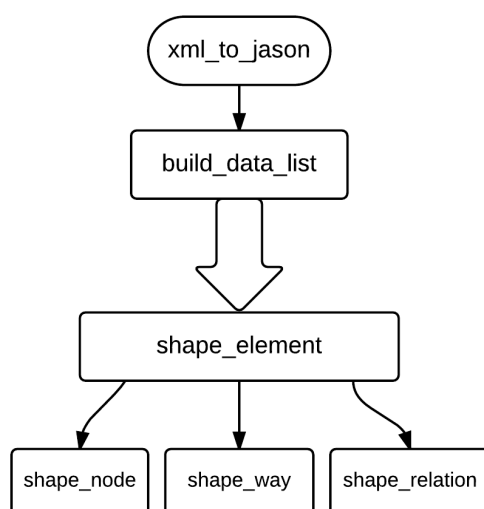
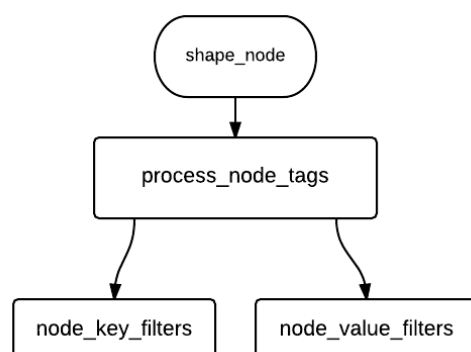# 1.2 Three Problems

Three problems are presented:

1. Problems associated with different type of keys of tag elements. Uniformity of keys makes search through Mongodb more easily. There are three procedures I have done to improve uniformity of keys. First, I find a handful chinese characters as keys in the xml. And I will discard these tag. Second, many keys with multiple words containing underscore("_") or hyphen("-"). I replace all the hyphens with underscore for uniformity of keys. Third, all the uppercase letters are replaced by lower case letters. These procedures are done in "xml_to_json.py".
2. Problem with colons in keys of tag element. As we can see from the result of running "audit.py". Some keys contain a single colon(":") and some keys contain double colons. Usually, colon means existence of sub-class. However, a smaller number of keys containing double colons which indicate level-two sub-class. I think level-one subclass is a standard way of dealing with these keys. I replace the second colon with underscore. And all these keys will only contain one colon. I construct sub-classes from these keys. For example, I have three keys with colon: "name:ch:simplified", "name:ch:traditional" and "name:en". These keys will form such structure: name_other:{ch_simplified:..., ch_traditional:..., en:...}. Noticing that I add "_other" suffix to make a new group of word. The procedure is also in "xml_to_json.py".
3. Problem of names. I noticed that some values of 'name' tag are neither Chinese character nor English words, such as "Guxiang 20" in the above output of "audit.py". These names are all pinyin which is a phonetic system of chinese characters. And these names are hardly used as names in Chinese. Luckily, these pinyins can be easily translated into more meaningful Chinese words. I replace all the pinyin in 'name' tag in the process. The procedure are also in "xml_to_json.py".
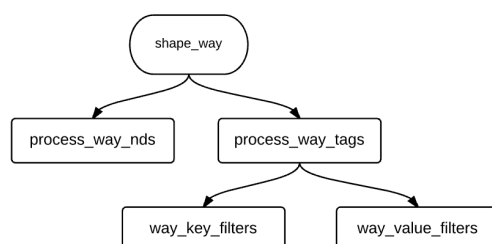
1.3 Structures of Cleaning Process

I will briefly introduce my structures of program which dealing with above problems. The following figure show the main structures of "xml_to_json.py".

```
        ┌──────────────┐
        │  xml_to_jason │
        └──────────────┘
                │
                ▼
        ┌──────────────┐
        │ build_data_list│
        └──────────────┘
                │
                ▼
        ┌──────────────┐
        │ shape_element │
        └──────────────┘
          │      │      │
          ▼      ▼      ▼
  ┌──────────┐┌──────────┐┌──────────────┐
  │shape_node││shape_way ││shape_relation│
  └──────────┘└──────────┘└──────────────┘
```
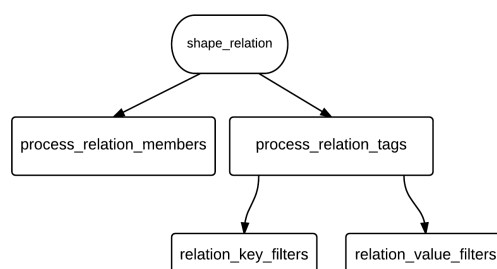
The program will build a list containing all the dictionaries from xml. The building process of this huge list is done by shaping individual element. There are three kinds of shaping method corresponding to three kinds of elements which is node, way and relation. The structure of shaping node is as following:

```
            ┌──────────────┐
            │  shape_node   │
            └──────────────┘
                    │
                    ▼
          ┌────────────────────┐
          │  process_node_tags  │
          └────────────────────┘
              │            │
              ▼            ▼
   ┌────────────────┐ ┌────────────────┐
   │ node_key_filters│ │node_value_filters│
   └────────────────┘ └────────────────┘
```

and way:

```
               ┌──────────┐
               │ shape_way │
               └──────────┘
                │        │
                ▼        ▼
    ┌────────────────┐ ┌────────────────┐
    │process_way_nds │ │process_way_tags│
    └────────────────┘ └────────────────┘
                        │            │
                        ▼            ▼
              ┌────────────────┐ ┌────────────────┐
              │ way_key_filters│ │way_value_filters│
              └────────────────┘ └────────────────┘
```

and relation:

```
               ┌──────────────┐
               │ shape_relation│
               └──────────────┘
                │            │
                ▼            ▼
 ┌──────────────────────┐ ┌──────────────────────┐
 │process_relation_members│ │ process_relation_tags │
 └──────────────────────┘ └──────────────────────┘
                            │            │
                            ▼            ▼
                ┌────────────────┐ ┌────────────────────┐
                │relation_key_filters│ │relation_value_filters│
                └────────────────┘ └────────────────────┘
```

In the function "process_node_tags", "process_way_tags" and "process_relation_tags", problem 2 and 3 are actually solved. Problem 1 are solved in "node_key_filters", "way_key_filters" and "relation_key_filters". The programs are structured such way to accommodate more possible data wranglings in future.

2. Data Overview

Data shaped by above process is imported into Mongodb. The json objects are insert into database "example" as collection "beijing_maps".

In [2]:

```
from db_functions import *
insert_maps(JSON_FILE, DB_NAME)
all_collections = get_collections(DB_NAME)
print JSON_FILE
print DB_NAME
```

```
../beijing_china.osm.json
examples
```

2.1 Size of xml file and json file

In [3]:

```
!du -h ../beijing_china.osm
!du -h ../beijing_china.osm.json
```

```
130M    ../beijing_china.osm
142M    ../beijing_china.osm.json
```

2.2 Some Total Counts

2.2.1 Total count of all the documents.

In [4]:

```
all_collections.find().count()
```

Out[4]:

699609

2.2.2 Total count of nodes, ways and relations.

In [5]:

```
all_collections.find({"type":"node"}).count()
```

Out[5]:

605204

In [6]:

```
all_collections.find({"type":"way"}).count()
```

Out[6]:

88958

In [7]:

```
all_collections.find({"type":"relation"}).count()
```

Out[7]:

5447

2.2.3 Counts associated with 'tag' element

Number of node, way and relation elements containing 'tag'.

In [8]:

```
all_collections.find({"type":"node", "tag":  {"$exists": 1}}).count()
```

Out[8]:

34490

In [9]:

```
all_collections.find({"type":"way", "tag":  {"$exists": 1}}).count()
```

Out[9]:

88240

In [10]:

```
all_collections.find({"type":"relation", "tag":  {"$exists": 1}}).count()
```

Out[10]:

5437

Number of node, way and relation elements containing 'name'.

In [11]:

```
all_collections.find({"type":"node", "tag.name":  {"$exists": 1}}).count()
```

Out[11]:

9610

In [12]:

```
all_collections.find({"type":"way", "tag.name":  {"$exists": 1}}).count()
```

Out[12]:

17124

In [13]:

```
all_collections.find({"type":"relation", "tag.name":  {"$exists": 1}}).cou
nt()
```

Out[13]:

4761

2.3 Some knowledge of updating time

Earliest time for updating of the map.

In [14]:

```
sorted_times = get_unique_time_sorted(all_collections)
print sorted_times[0]
```

2007-03-14 18:09:10

Latest time when the map file is downloading.

In [15]:

```
print sorted_times[-1]
```

2015-08-28 14:04:24

The time when people update most frequently on this map.

In [16]:

```
pipeline = [{'$group':{'_id':'$timestamp',
                        'count':{'$sum':1}}},
            {'$sort':{'count':-1}},
            {'$limit':1}
            ]
list(all_collections.aggregate(pipeline))
```

Out[16]:

```
[{u'_id': u'2015-06-12T06:31:41Z', u'count': 92}]
```

3. Additional Ideas

3.1 Inconsistency of Languages and Localization of Map Data

I will take names of nodes and ways as example. The names in the map of Beijing should be in Chinese character instead of pinyin which is the official phonetic system of the Mandarin. Pinyin is meaningless in writing or printing the Mandarin. However, I find many names of pinyin in the map. In other words the names are not consistent.

In [18]:

```
from process_util import *
result = read_word_list(NODE_NAME_FILE)
print 'Total number of names of nodes is '+ str(len(result)) + '.'
print 'Number of pinyin is ' + str(count_map(pinyin_reg, result)) + '.'

result = read_word_list(WAY_NAME_FILE)
print 'Total number of names of ways is ' + str(len(result)) + '.'
print 'Number of pinyin is ' + str(count_map(pinyin_reg, result)) + '.'
```

```
Total number of names of nodes is 5822.
Number of pinyin is 988.
Total number of names of ways is 7724.
Number of pinyin is 721.
```

Based on this example, We can see that some nodes is named in chinese and some in english or pinyin which is not a language. Besides the problem of different languages in names. The most key of tag elements which are most important source of information in the XML file is in english. I think inconsistency of languages prevent the development of Openstreetmap in non-english country. And not all the names of entities or categories can properly represented in English.

Localization of map data can be implemented by changing structure of data. The tag element could contain multiple sub-elements corresponding to different languages of description for that tag. The names of these sub-elements are the standardized names of the languagea used in both key and value attributes in the corresponding sub-element. The kind of language the tag used can be corporated into data collecting process of OpenStreetMap or examined by a program which can easily identify many different languages.

However, the modification of data structure must be carried out carefully. First, the new version of data may incompatible with programs which deal with older version of data. This result may effect thousands of user who already use OpenStreetMap. Second, adding language information of tag element introduces more complexity of data. Cleaning of data and processing data will be more complicated than older version. Third, more data is needed for describing a map. Because older version of data blend multiple languages to deliver information. If the information is divided into different languages. The information may loss in single language.

3.2 Commit history of the map.

First, let's see the varying of total numbers of updates according to dates. The updating counts keeps growing before year 2012. However, after a sudden increasing in 2012, the growing stopped. The image is produced by script in db_functions.py using Mongodb query and matplotlib.
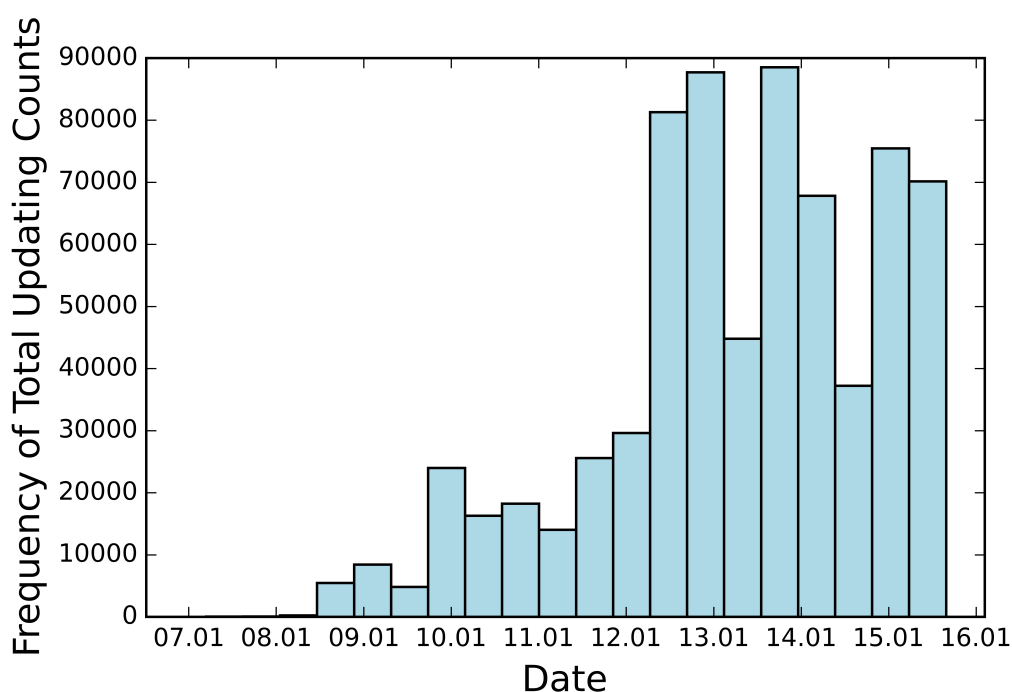
One may ask, why does the counting of updating grow like this way. Did the number of users of Openstreetmap stopped to increase since 2012? Let us first look at the distribution of updating counts for different users. The distribution is highly skewed. Most users only updating less a few times. And most of updates are commited by a handful users. Total number of users is 1102. And almost 90% user updating less than 100 times.The image is produced by script in db_functions.py using Mongodb query and matplotlib.
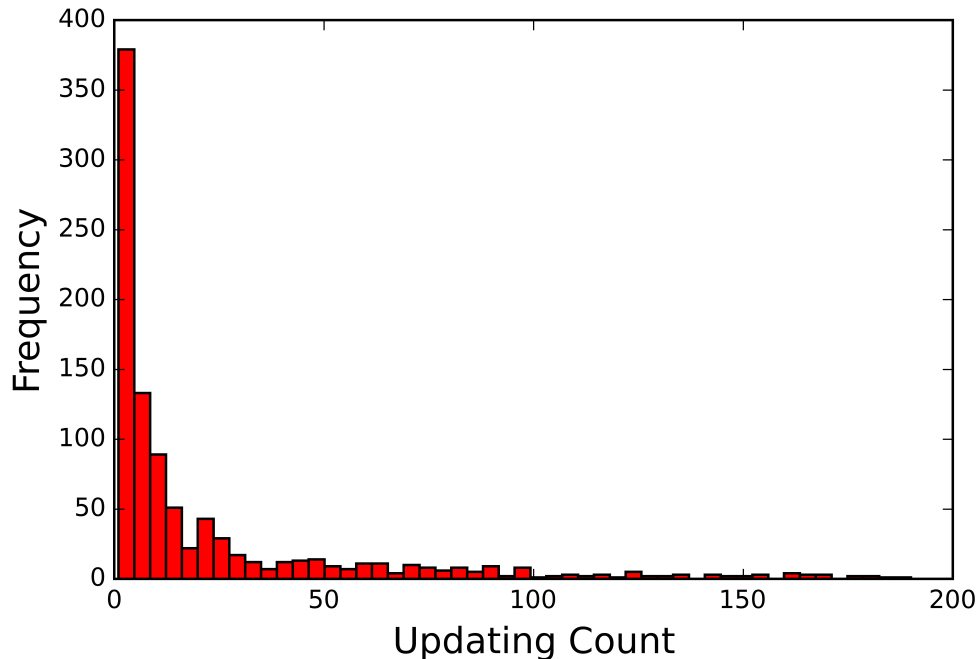
Majority of users updating less than 100 times. I will call this part of users "Lower updating users". The rest of user is "higher updating users". Since most users are lower updating users. I will plot how contribution of lower updating users evolved acrossing time. Following is the figure. The figure shows that during last 8 years, the counting is consistently increasing. I believe that this is the evident which shows the Openstreetmap is making significant progress during last few years. The image is produced by script in db_functions.py using Mongodb query and matplotlib.
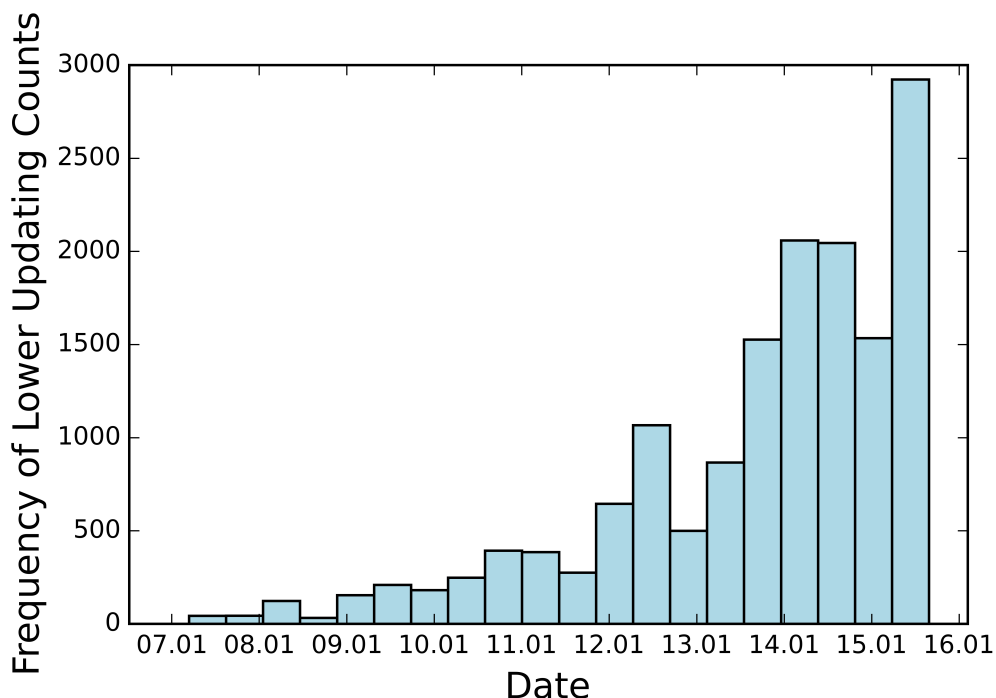
But how the sudden change of updating counts happened in year 2012? The plot of higher updating users reveals the answer. The plot is almost identical to the total counting trend. Because of the participation of some higher updating users in year 2012 the counting is increasing dramatically at that time point. The image is produced by script in db_functions.py using Mongodb query and matplotlib.
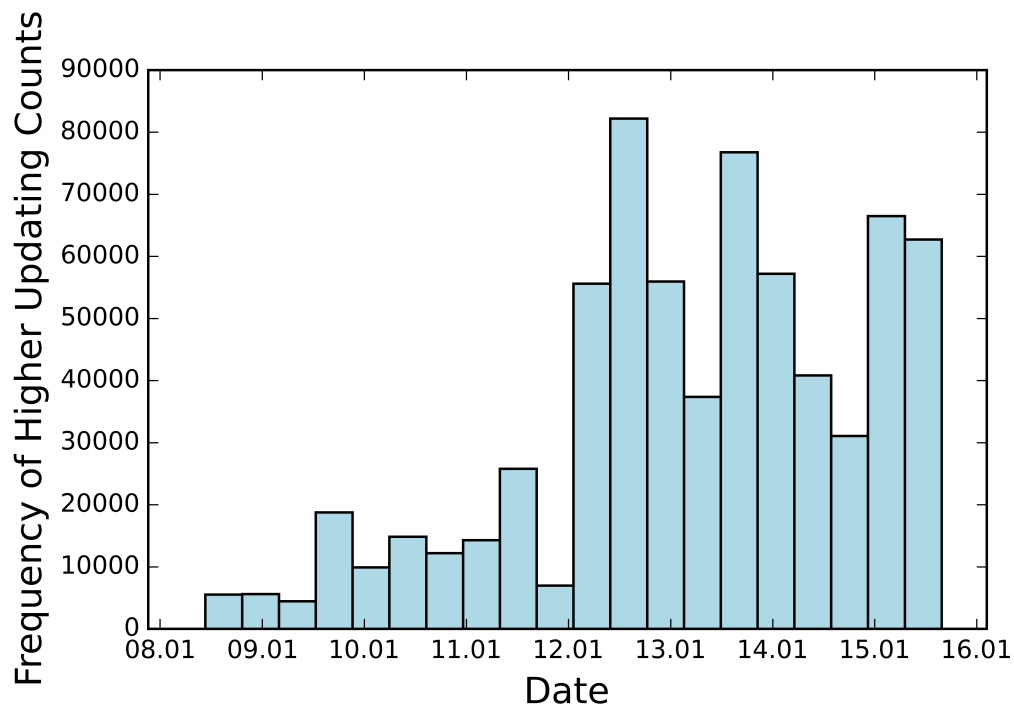
4. Conclusion

The map data of Beijing from Openstreetmap is carefully examined and summarized. Important features such as scale, user, updating time and other statistical numbers were given by this article. Several problems associated with this dataset are proposed and most of them are solved in a systemic way. These problems include inconsistency of keys, irregularity of words and improper names of entities. I observed the map data is far from completeness. The map lack a lot of descriptions of nodes especially. And I also noticed that many names in the data set is in pinyin which is not a right way to identify map entities. Some commit history is examined and showed to verify the fact that the user of Openstreetmap is in a consistent growing.