

Wrangle OpenStreetMap Data in MongoDB

Map Area:

Beijing,China

<https://www.openstreetmap.org/relation/912940>

<https://mapzen.com/data/metro-extracts/>

Beijing is my hometown. I'm familiar with the city and I can easily find out the error in xml data. I also like to discover what the map database reveals.

Problems Encountered in the Map

I initially downloaded 10% of osmxml data of Beijing and run it against data_mongo.py; I noticed there were some problems with the data:

1. Nodes name tags like `<tag k="name:af" v="Beijing" />`, `<tag k="name:an" v="Pequín" />` are many language versions and they should be converted as `{"name": {"af": "Beijing", "an": "Pequín" ...}}`.
2. Address Postcode Tags like `<tag k="addr:postcode" v="10043" />` have error value like "10043", all postcode should be 6 digit numbers begin with "1".
3. Address City Tags like `<tag k="addr:city" v="北京市海淀区" />` have values should be "北京", "北京市" or "Beijing", "beijing", there're some error city values like "北京市海淀区","怀柔区雁栖镇"
4. Roof Colour Tags like `<tag k="roof:colour" v="orange" />` ,have values should be hex code begin with "#", some error values are colors : "orange","grey","green","red","brown","blue","gray", "black","white"

Processing the problems

1. Correcting Name tags

I. Name tags like `<tag k="name:af" v="Beijing" />`, `<tag k="name:an" v="Pequín" />` be converted as `{"name": {"af": "Beijing", "an": "Pequín" ...}}`, the same processing strategy with 'addr:'. For "name" without following ":" like `<tag k="name" v="统军庄新路口" />`, they are converted as `{"name": "name"}`. Here's my code in data_mongo.py:

```
for tag in element.iter("tag"):
    if re.match(problemchars,tag.attrib['k'])==None:
        if tag.attrib['k'].find('.')>=0:
            if tag.attrib['k'].find('addr:')>=0:
```

```

        if tag.attrib['k'][tag.attrib['k'].find(':')+5:].find(':')<0:

address[tag.attrib['k'][tag.attrib['k'].find('addr:')+5:]=tag.attrib['v']

        elif tag.attrib['k'].find('name:')>=0:
            if tag.attrib['k'][tag.attrib['k'].find(':')+5:].find(':')<0:

name[tag.attrib['k'][tag.attrib['k'].find('name:')+5:]=tag.attrib['v']

        else:

            node[tag.attrib['k']]=tag.attrib['v']

    else:
        if tag.attrib['k']=='name':
            name['name']=tag.attrib['v']
            node[tag.attrib['k']]=tag.attrib['v']
        if name!={}:
            node['name']=name
        if address!={}:
            node['address']=address

```

2. Correcting Postcode

I found several postcodes were wrong with 5 numbers, such as "10043","10080",I corrected them from 100XX to 1000XX. Here's my code in osm_bj_1_mongodb.py:

```

postcode=db.beijingfull.find({'address.postcode':{'$regex':'^[0-9]{0,5}$'}},{'address.postcode':1,'id':1})
for p in postcode:
    print "id{} postcode {} not correct".format(p['id'],p['address'])
    if p['address']['postcode'].find('100')>=0:
        p['address']['postcode']=p['address']['postcode'].replace('100','1000')
        db.beijingfull.save(p)
        print 'id{} postcode is corrected to {}'.format(p['id'],p['address'])

```

Note that postcode not in 100XX format can't be corrected. There's one postcode with 4 digits can't be corrected.

3.Processing Errors in tags key="addr:city"

I replace all noncity values with 'BEIJING' though code in osm_bj_1_mongodb.py:

```
element=db.beijingfull.find({'$and':[{'address.city':{'$ne':'beijing'}},{'address.city':{'$ne':'Beijing'}},{'address.city':{'$exists':1}},{'address.city':{'$ne':'北京'}},{'address.city':{'$ne':'北京市'}}]},{'_id':0,'id':1,'address.city':1})
print 'address.city need to correct:\n'
for e in element:
    pprint (e)
db.beijingfull.update({'$and':[{'address.city':{'$ne':'beijing'}},{'address.city':{'$ne':'Beijing'}},{'address.city':{'$exists':1}},{'address.city':{'$ne':'北京'}},{'address.city':{'$ne':'北京市'}}]},{'$set':{'address.city':'BEIJING'}},multi=True)
print "address:city have been corrected to 'BEIJING'."
```

4.Processing Errors in tags key= "roof:colour"

I replace all error roof:colour values with correct color hex from: <http://www.colorhexa.com/>

At first trial with k=10(10% sampling), I found error color values of "orange""grey""green""red", and at the trial with k=5(20% sampling), I added "brown""blue""gray", and finally trial with full dataset, I added 'black'white'.


code in osm_bj_1_mongodb.py:

```
db.beijing5.update({'roof:colour':'red'},{'$set':{'roof:colour':'#ff0000'}},multi=True)
print "roof:colour red have been corrected."
db.beijing5.update({'roof:colour':'orange'},{'$set':{'roof:colour':'#ffa500'}},multi=True)
print "roof:colour orange have been corrected."
db.beijing5.update({'roof:colour':'green'},{'$set':{'roof:colour':'#00ff00'}},multi=True)
print "roof:colour green have been corrected."
db.beijing5.update({'roof:colour':'blue'},{'$set':{'roof:colour':'#0000ff'}},multi=True)
print "roof:colour blue have been corrected."
db.beijing5.update({'roof:colour':'brown'},{'$set':{'roof:colour':'#a52a2a'}},multi=True)
print "roof:colour brown have been corrected."
db.beijing5.update({'roof:colour':'black'},{'$set':{'roof:colour':'#000000'}},multi=True)
print "roof:colour black have been corrected."
db.beijing5.update({'roof:colour':'white'},{'$set':{'roof:colour':'#ffffff'}},multi=True)
print "roof:colour white have been corrected."
db.beijing5.update({'$or':[{'roof:colour':'grey'},{'roof:colour':'gray'}]},{'$set':{'roof:colour':'#808080'}},multi=True)
print "roof:colour grey and gray have been corrected."
```

Data Processing

Download original data file

Here's the full osm xml file

 beijing_china.osm	2016/7/27 11:13	OSM 文件	154,615 KB
---	-----------------	--------	------------

Run data_mongo.py to get the json

JSON file after processing by data_mongo.py

 beijing_china.osm.json	2016/8/8 17:25	JSON 文件	234,458 KB
--	----------------	---------	------------

Import json the mongodb

```
C:\Program Files\MongoDB\Server\3.2\bin>mongoimport -d osm -c beijingfull --file
beijing_china.osm.json
2016-08-08T17:28:40.718+0800    imported 815954 documents
```

Run osm_bj_1_mongodb.py to correct errors:

Data Overview and Additional Ideas

Data statistics:

Number of nodes

```
db.beijingfull.find({'type':'node'}).count()
710867
```

Number of ways

```
db.beijingfull.find({'type':'way'}).count()
105072
```

Number of unique users

```
len(db.beijingfull.distinct('created.uid'))
1409
```

Top 10 contributing users

```
pipeline=[
{'$group':{'_id':{'uid':'$created.uid','name':'$created.user'},'count':{'$sum':1}},
{'$sort':{'count':-1}},
```

```
{'$limit':10}}
top10user=db.beijingfull.aggregate(pipeline)
```

Top 10 contributing users:

```
{'u'count': 194009, 'u'_id': {'u'uid': 'u'288524', 'u'name': 'u'Chen Jia'}}
{'u'count': 151236, 'u'_id': {'u'uid': 'u'376715', 'u'name': 'u'R438'}}
{'u'count': 52067, 'u'_id': {'u'uid': 'u'139957', 'u'name': 'u'ij_'}}
{'u'count': 47776, 'u'_id': {'u'uid': 'u'499500', 'u'name': 'u'hanchao'}}
{'u'count': 24052, 'u'_id': {'u'uid': 'u'17497', 'u'name': 'u'katpatuka'}}
{'u'count': 21995, 'u'_id': {'u'uid': 'u'486052', 'u'name': 'u'm17design'}}
{'u'count': 19065, 'u'_id': {'u'uid': 'u'83557', 'u'name': 'u'Esperanza36'}}
{'u'count': 17230, 'u'_id': {'u'uid': 'u'75424', 'u'name': 'u'nuklearerWintersturm'}}
{'u'count': 14490, 'u'_id': {'u'uid': 'u'2639622', 'u'name': 'u'RationalTangle'}}
{'u'count': 9411, 'u'_id': {'u'uid': 'u'421504', 'u'name': 'u'u_kubota'}}
```

Number of users appearing least(283 users only have 1 post)

```
pipeline=[
{'$group':{'_id':{'uid':'$created.uid','user':'$created.user'},'count':{'$sum':1}},
{'$group':{'_id':'$count','num_users':{'$sum':1}}},
{'$sort':{'_id':1}},
{'$limit':1}
]
num1post=db.beijingfull.aggregate(pipeline)
```

Number of users appearing least
{'u'num_users': 283, 'u'_id': 1}

Additional Ideas:

1. Additional Data Exploration

I explored the top appearing amenities. In Beijing the most appearing amenity is restaurant. The most popular cuisine of restaurant is Chinese. Japanese and Italy food are also very popular. I also have a trial to use \$near and \$maxDistance to get the nearest restaurants in radians=200m of Capital Airports [116.5849695,40.0711596](<lon>,<lat>). The Beijing_China OSM has been created since 2007 and 14.6% documents created in 2016(Jan-July), 17.7%documents created in 2015. There're 10 years since created the first document in 2007. The average growth is 10%. In last 2 years, the data growth of 2015 and 2016 both beyond the average growth of 10%. That means the city is growing and the OSM is growing both with the city.

Top 10 appearing amenities

```
code in osm_bj_1_mongodb.py:
pipeline=[
{'$match':{'amenity':{'$exists':1}}},
```

```
{ "$group": { "_id": "$amenity", "count": { "$sum": 1 } },
  { "$sort": { "count": -1 } },
  { "$limit": 10 }
]
```

amenities=db.beijingfull.aggregate(pipeline)

result:

top 10 appearing amenities:

```
{'count': 1047, '_id': 'restaurant'}
{'count': 615, '_id': 'parking'}
{'count': 371, '_id': 'bank'}
{'count': 354, '_id': 'school'}
{'count': 338, '_id': 'toilets'}
{'count': 278, '_id': 'fuel'}
{'count': 258, '_id': 'fast_food'}
{'count': 185, '_id': 'cafe'}
{'count': 156, '_id': 'hospital'}
{'count': 152, '_id': 'telephone'}
```

Most popular cuisines:

code in osm_bj_1_mongodb.py:

```
pipeline=[
  { '$match': { 'amenity': { '$exists': 1 }, 'amenity': 'restaurant', 'cuisine': { '$exists': 1 } },
  { '$group': { '_id': '$cuisine', 'count': { '$sum': 1 } },
  { '$sort': { 'count': -1 } },
  { '$limit': 3 }
]
```

cuisine=db.beijingfull.aggregate(pipeline)

result:

most popular cuisines:

```
{'count': 119, '_id': 'chinese'}
{'count': 12, '_id': 'japanese'}
{'count': 10, '_id': 'pizza'}
{'count': 10, '_id': 'regional'}
{'count': 9, '_id': 'italian'}
```

The nearest 10 restaurants to ‘Capital Airport’

Query in mongo console:

```
> db.beijingfull.find ( { 'pos' : { $near : [116.5849695,40.0711596], $maxDistance:
  200 }, 'amenity': 'restaurant', 'name': { $exists: 1 }, { '_id': 0, 'amenity': 1, 'name': 1,
  'pos': 1 } }
{ "amenity" : "restaurant", "name" : { "name" : "Pinnacle Plaza" }, "pos" : [ 11
6.5412853, 40.0614854 ] }
{ "amenity" : "restaurant", "name" : { "name" : "大掌勺" }, "pos" : [ 116.645611
4, 40.1141037 ] }
{ "amenity" : "restaurant", "name" : { "name" : "顺宜坊" }, "pos" : [ 116.648925
2, 40.111247 ] }
```

```

{"amenity": "restaurant", "name": {"name": "福成肥牛(火锅)", "pos": [ 11
6.6502815, 40.1107706 ] } }
{"amenity": "restaurant", "name": {"name": "全聚德", "pos": [ 116.63464,
40.1342102 ] } }
{"amenity": "restaurant", "name": {"name": "金佰万", "pos": [ 116.641309
1, 40.1285157 ] } }
{"amenity": "restaurant", "name": {"name": "上一档", "pos": [ 116.656838
, 40.1080809 ] } }
{"amenity": "restaurant", "name": {"name": "久久嘉", "pos": [ 116.658357
8, 40.1070672 ] } }
{"amenity": "restaurant", "name": {"name": "红菜坊", "pos": [ 116.659443
9, 40.1068962 ] } }
{"amenity": "restaurant", "name": {"name": "比格皮萨顺义店", "pos": [ 11
6.6458589, 40.1276162 ] } }
{"amenity": "restaurant", "name": {"name": "呷哺呷哺 NO.218", "pos": [ 11
6.6457338, 40.1277901 ] } }
{"amenity": "restaurant", "name": {"name": "创新高", "pos": [ 116.660025
, 40.1107311 ] } }
{"amenity": "restaurant", "name": {"zh": "安妮", "name": "Annie's", "pos
": [ 116.5050102, 40.0232652 ] } }
{"amenity": "restaurant", "name": {"name": "大厨房", "pos": [ 116.658891
3, 40.1352231 ] } }
{"amenity": "restaurant", "name": {"name": "IOWA", "pos": [ 116.5050901,
39.9933388 ] } }
{"amenity": "restaurant", "name": {"name": "Wine Talks", "pos": [ 116.50
19465, 39.9922912 ] } }
{"amenity": "restaurant", "name": {"name": "Maan Coffee", "pos": [ 116.5
047738, 39.971998 ] } }
{"amenity": "restaurant", "name": {"name": "那家小馆", "pos": [ 116.4876
527, 39.9863056 ] } }
{"amenity": "restaurant", "name": {"name": "金百万", "pos": [ 116.472021
5, 40.0072536 ] } }
{"amenity": "restaurant", "name": {"name": "BenJia", "pos": [ 116.472990
1, 40.0054086 ] } }

```

Type "it" for more

How many documents created in 2016 and 2015.

My code in osm_bj_1_mongodb.py:

```
year=Set([])
```

```
element=db.beijingfull.find({'created.timestamp':{'$exists':1}})
```

for e in element:

```
    year.add(e['created']['timestamp'][0:e['created']['timestamp'].find('-')])
```

```
print year
```

```
print '{} total docs from 2007'.format(db.beijingfull.find().count())
```

```

print          '{}'          docs          created          in
2016'.format(db.beijingfull.find({'created.timestamp':{'$gte':'2016-01-01T00:00:00Z'}}).count())
print          '{}'          docs          created          in
2015'.format(db.beijingfull.find({'created.timestamp':{'$gte':'2015-01-01T00:00:00Z','$lte':'2015-
12-31T23:59:59Z'}}).count()))
result:
Set([u'2009', u'2007', u'2015', u'2014', u'2008', u'2016', u'2011', u'2010', u'2
013', u'2012'])
815954 total docs from 2007
119271 docs created in 2016
144674 docs created in 2015

```

2. Suggestions for Improving OSM dataset

The contributions of users seem obviously skewed. The top 2 users contributed 48.6% nodes and ways data. 20% users only contribute 1 post.

- Top user contribution percentage (“Chen Jia”) 27.3%
- Combined top 2 users' contribution (“Chen Jia” and “R438”) 48.6%
- Combined Top 10 users contribution 77.5%
- The number of users making up only 1 post is 283(20% of total users).

Top 10 contributing users:

```

{'u'count': 194009, 'u'_id': {'u'uid': 'u'288524', 'u'name': 'u'Chen Jia'}} contributed 27.3%
{'u'count': 151236, 'u'_id': {'u'uid': 'u'376715', 'u'name': 'u'R438'}} contributed 21.3%
{'u'count': 52067, 'u'_id': {'u'uid': 'u'139957', 'u'name': 'u'ij_'}}
{'u'count': 47776, 'u'_id': {'u'uid': 'u'499500', 'u'name': 'u'hanchao'}}
{'u'count': 24052, 'u'_id': {'u'uid': 'u'17497', 'u'name': 'u'katpatuka'}}
{'u'count': 21995, 'u'_id': {'u'uid': 'u'486052', 'u'name': 'u'm17design'}}
{'u'count': 19065, 'u'_id': {'u'uid': 'u'83557', 'u'name': 'u'Esperanza36'}}
{'u'count': 17230, 'u'_id': {'u'uid': 'u'75424', 'u'name': 'u'nuklearerWintersturm'}}
{'u'count': 14490, 'u'_id': {'u'uid': 'u'2639622', 'u'name': 'u'RationalTangle'}}
{'u'count': 9411, 'u'_id': {'u'uid': 'u'421504', 'u'name': 'u'u_kubota'}}

```

I think the most important thing to OSM is the volunteers constantly actively contribute data to OSM. OSM is considered a prominent example of volunteered geographic information (VGI). VGI offers the possibility of including subjective, emotional, or other non-cartographic information. That's the advantage in contrast to traditional map. I think one of very effective ways to improve OSM data is to cooperate with geo referenced applications and services such as TripAdvisor, Pokemon Go etc. These services and applications can use map data of OSM, users can add comments, or likes to the nodes on OSM. And further, additional services can be provided to user base on the user-generated contents. For example, recommending the neighbor restaurants with cuisines, likes and comments to map users. The usages of OSM map can popularize OSM and deeply encourage individual users add and modify data in OSM actively, both geo data and

subjective data. This will be a good mechanics to help OSM grow with the city together and become a handy tool for city daily life.

The problem may also introduced into implement improving is how to check and guarantee the data quality. I think there may be some good experience and best practices of Wikipedia or other user-generated content examples to learn from. How to display users' subjective comments is also a challenge to face to. Should bad comments be displayed in recommending? Or only display neutral characters without emotional comments? That's to think about before implement applications based on user-generated contents.

Conclusion:

In the data wrangling of OpenStreetMap Beijing_China xml data, I found that the data quality is high. Data in the last 2 years rapidly grows above average growth rate. But still far from completed. I can easily found the nodes and ways are not included in the dataset. I also have discussed and provided suggestions for improving OSM data. That is cooperation with geo referenced services and applications to popularize usage of OSM and provide geo related user-generated contents to users. The more usage of OSM and popularity will encourage more individuals to contribute map data and subjective data actively. Problems will be introduced is how to guarantee the user generated data quality, and how to use the subjective user-generated contents properly.