Udacity: Wrangle OpenStreetMap Data with SQL

Siye Yu

07/20/2017

Background:

OpenStreetMap (OSM) is a open source map database. Due to the natural of user generated data, error is unavoidable in its data. In this project, I will try to make some corrections to its data in one particular metropolitan area. The following steps will answer questions in the project rubrics.

In addition to the submission, I will further improve on the corrected data and submit it to OSM to improve the mapping data.

Step 1 – Download OSM Data:

I picked "Shanghai, China" as the metropolitan area due to its size and relative smaller user base, in anticipation of possible errors and inconsistency.

Download link:

https://mapzen.com/data/metro-extracts/metro/shanghai_china/

The extracted file size is over 700mb, which is still too big to process. We need to generate a smaller sample for testing purpose.

Step 2 – Generate Sample Data:

Based on the code from the project page (https://classroom.udacity.com/nanodegrees/nd002/parts/0021345404/modules/316820862075463/lessons/3168208620239847/concepts/77135319070923), and choose "k = 100" for a file size about 1/100th of original file size.

Please refer to osm.py #section 1 for complete code.

Now I have a working sample file – shanghai_sample.osm, which will be used throughout the project.

Step 3 – Data Basics

Let's take a look at the raw data for a better insight. First of all, I counted tags of the data. (Please refer to #section 2 of osm.py for code)

Result:

```
{'member': 687,
 'nd': 41728,
 'node': 35559,
 'osm': 1,
 'relation': 32,
 'tag': 12488,
 'way': 4457}
```

Now we have a good size of elements to work with.

Next step, let me take a look to see if there's any anomaly in each tag's k value, which is prone to errors. (Please refer to #section 3 of osm.py for code)

Result:

```
{'lower': 111046, 'lower_colon': 11884, 'other': 269, 'problemchars':
0}
```

Great, there's no problem characters in the data. But I do see a large amount of lower_colon in the data.

I also notice, the tag "name:en" has a large quantity in the data. Since the data is about a city in China, it is natural to have the English name alongside the Chinese name. Let's dig deeper to see possible problems with this attributes.

Step 4 – Find errors:

Optional: Print out all the English names to see possible errors. (Please refer to #section 4 of osm.py for code) This can be useful for small file size, but not recommended for large file.

Let's find all the possible ending of English name to look for error patterns. Immediately, several error types is easy to spot.

Result:

```
'Hwy.': {'Caoan Hwy.',
         'Dachuan Hwy.',
         'Fenghua Hwy.',
         'Fenghuang Hwy.',
         'Hutai Hwy.',
         'LiuFeng Hwy.',
```

```
           'Liuxiang Hwy.',
           'Nantuan Hwy.',
           'Xiangjiang Hwy.'},
'Rd.': {'Caobao Rd.',
          'Hexuan Rd.',
          'Huajiang Branch Rd.',
          'Jinshi Rd.',
          'Luwu Rd.',
          'Shuishan Rd.',
          'Taopu Rd.',
          'Tingchao Rd.',
          'Wudu Rd.'},
 'S)': {'Lintao Rd. (S)'},
```

The abbreviation of “Rd.”, “Hwy.”, and the “(S)/(N)/(W)/(E)” errors. For example, ‘'Lintao Rd. (S)’ should be written as ‘South Lintao Road’.

Step 5 – Correct Errors:

After analyze the error patterns, I wrote a program to make corrections. Please refer to #section 5 of osm.py for code). With some testing, the code seems to be working fine.

Partial result:

```
Fenghua Hwy. => Fenghua Highway
Nantuan Hwy. => Nantuan Highway
Xiangjiang Hwy. => Xiangjiang Highway
Hutai Hwy. => Hutai Highway
Liuxiang Hwy. => Liuxiang Highway
Wudu Rd. => Wudu Road
Caobao Rd. => Caobao Road
Luwu Rd. => Luwu Road
Taopu Rd. => Taopu Road
Tingchao Rd. => Tingchao Road
Shuishan Rd. => Shuishan Road
Hexuan Rd. => Hexuan Road
Huajiang Branch Rd. => Huajiang Branch Road
Jinshi Rd. => Jinshi Road
Baozhen Rd. (N) => North Baozhen Road
Jiasong Rd. (N) => North Jiasong Road
Lintao Rd. (S) => South Lintao Road
Zhenxing Rd. (E.) => East Zhenxing Road
```

Next I’ll combine the name correction with separate different element into separated csv files.

Step 6 – Generate CSV Files:

With code from previous lessons, plus my tweaks. I successfully generate the following csv files. Please refer to #section 6 of osm.py for code.

```
['nodes.csv', 'nodes_tags.csv', 'ways.csv', 'ways_nodes.csv',
'ways_ta gs.csv']
```

Everything looks good. Let's turn them into database.

Step 6B – SQL

Checking database osm.db in sqlite to gather some information about the database

```
SELECT value, COUNT(*) as num
FROM nodes_tags
WHERE key='amenity'
GROUP BY value
ORDER BY num DESC
LIMIT 10;

bicycle_rental|26
restaurant|20
bank|5
bus_station|5
toilets|4
bar|3
cafe|3
community_centre|3
fuel|3
parking|3
```

Not surprising, with the booming bike sharing startup in China. Bibycle_rental has become a popular amenity.

```
sqlite> SELECT COUNT(*) FROM nodes;
35559
sqlite> SELECT COUNT(*) FROM ways;
4457
sqlite> SELECT COUNT(DISTINCT(e.uid))
   ...> FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways)
e;
1081
```

Look up file size in MacOS terminal

```
du -h *.db
4.4M  osm2.db
```

Step 7 – Generate Database file:

Following the #section 7 of osm.py. I successfully generated a db file based on the csv files.

Additional Improvement Discussion:

The file still contains many errors, there are large portion of tags with only Chinese names, also some tags only has English names, and some with multiple name tags which can be confusing for users. It can be a good idea to write a function which will translate these names from google translate API. And split English and Chinese name into prospect tag. For example, same name tag contains both English and Chinese:

```
'江南公路 Jiangnan Expressway',
'王府饭店 Wangfu Hotel',
```

Please refer to #section 9 of osm.py for additional error sample code.

Potential improvement issue:

With fully corrected English name alongside Chinese name, it will be beneficial to foreign tourists when using OPM inside China. Due to China Authoritarian government's censorship policy, Google Map cannot function inside China properly. And local map may be confusing for foreigner to navigate.

But Google Translate may not return the corrected/official English name, and errors can be hard to track down with hundreds of thousands of records. It is possible to use human audit, but it defeats the purpose of the data project.

There is no perfect solution. I'll do my best to continue the DAND course, and maybe utilize Machine Learning to solve this problem in future.

Additional Question:

Regarding previous review's comment on code:

> This is the current `shape_element` function's definition:
>
> ```
> def shape_element(element, node_attr_fields=NODE_FIELDS,
> way_attr_fields=WAY_FIELDS,
> problem_chars=PROBLEMCHARS, default_tag_type='regular'):
> ```
>
> Currently, these four parameters `node_attr_fields` to `default_tag_type` was never used in the code. Instead, the constants were used e.g. `NODE_FIELDS` instead of `node_attr_fields`. This issue did not cause the functions to stop working properly, so I do not mark this as failing the specification, but I strongly suggest to replace the constants with parameter variables so you can use the functions more flexibly in the future.

Does that mean I should remove NODE_FIELDS, WAY_FIELDS, and use node_attr_fields as variable? In order to make this more functionable in the future?