Thursday, January 12, 2017

Project 3: OpenStreetMap Data Wrangling Project

Sheena YU

Summary

In this project, I'm going to explore an OpenStreetMap dataset which focuses on Dallas,TX area. I've been living in Dallas for one year where I finished my Master's degree in Southern Methodist University. There're many places I haven't got a chance to visit so I think it is a wonderful metropolitan to explore through the data wrangling techniques I gained.

Source of Dallas map data (OSM XML)

Part 1: Problems Encountered in the Map

- **Redundant Elements**

`mapparser.py` is used for parsing the OSM XML file. The first step I take is to identify the elements as well as their attributes in this XML file and choose the ones I need for further analysis.

According to the results from `mapparser.py` , I extracted the following elements' tags and the frequency of their occurences:

```
[('nd', 3155335),
 ('node', 2711877),
 ('tag', 2377036),
 ('way', 310384),
 ('member', 28508),
 ('relation', 1850),
 ('osm', 1),
 ('bounds', 1)]
```

Among these elements, `node` and `way` are what I'm interested in keeping to upload to the database. From [Wiki OpenStreetMap](https://wiki.openstreetmap.org/wiki/Elements), nodes represent specific points on the earth's surface defined by their latitudes and longitudes, while ways define linear features(rivers, roads, etc.) and area boundaries(buildings, forests, etc.).

A relation is a multi-purpose data structure that documents a relationship between two or more data elements. Thus, relations can have different meanings. The other elements are either children of the node, way, member and relation elements, or specialized elements.

For best serving the purpose of analyzing the node and way elements, I decide to parse other elements out.

- **Unexpected k attributes**

For `k` attributes in child elements whose tag name is 'tag', I extracted the most occurred k attributes with frequency greater than 4,000 out of these elements.

```
('highway', 266943), ('tiger:county', 175891), ('tiger:cfcc', 175461),
('name', 167182), ('tiger:reviewed', 159595), ('tiger:name_base',
119025), ('tiger:source', 118154), ('tiger:tlid', 117717),
('tiger:separated', 105648), ('tiger:name_type', 104402),
('tiger:zip_left', 77965), ('tiger:zip_right', 72834)
('addr:housenumber', 58644), ('addr:street', 58338), ('addr:city',
57072)('tiger:upload_uuid', 51602), ('addr:full', 49520), ('oneway',
44996), (`building',32012), ('power', 22968), ('service', 19291),
('lanes', 17303), ('tiger:name_direction_prefix', 16675), ('amenity',
14289),('ref', 13071), ('access', 12388), ('tiger:name_base_1',11972),
('source', 9041), ('ele', 7662), ('layer', 7478),(`gnis:feature_id',
7186), ('railway', 6958), ('bridge', 6905),('gnis:created', 6602),
('gnis:county_id', 6341),('gnis:state_id',6339), ('maxspeed', 6104),
('turn:lanes',5841), ('landuse', 5160),('name_1', 5136), ('surface',
5071),('addr:postcode', 5058),('old_ref', 4860), ('leisure', 4609),
('natural', 4562)…
```

Surprisingly, I found many k attributes containing "tiger: ***" and "gnis: ***". I searched online and figured out that these are different data systems which document geographic features. Since `GNIS` don't contribute much useful information in our map data, I chose to just keep some [TIGER](http://wiki.openstreetmap.org/wiki/TIGER) values.

- **Complicated TIGER Geographic Attributes**

After doing a thorough research, I created a “tiger” dictionary for each node that contains tiger information, and within the dictionary I choose to keep these keys: `street_name`, `county`, `zipcode` and `cfcc`.

“cfcc” stands for Census Feature Class Codes. Please check out the complete “cfcc” lookup table.

For example:
The original tags containing "tiger" information are listed below:

```
<tag k="tiger:cfcc" v="A41" />
<tag k="tiger:tlid" v="101893004" />
<tag k="tiger:county" v="Tarrant, TX" />
<tag k="tiger:source”v="tiger_import_dch_v0.6_20070830" />
<tag k="tiger:reviewed" v="no" />
<tag k="tiger:zip_left" v="76104" />
<tag k="tiger:name_base" v="Morningside" />
<tag k="tiger:name_type" v="Dr" />
<tag k="tiger:separated" v="no" />
<tag k="tiger:zip_right" v="76104" />
<tag k="tiger:name_direction_prefix" v="E" />
```

After transformation, the "tiger" dictionary will look like this:

```
'tiger': {'county': 'Tarrant, TX',
          'street_name': 'East Morningside Drive',
          'cfcc': 'A41',
          'zipcode': [‘76104']}
```

- **Already-existing "type" Attribute**

In the data wrangling phase, I added a "type" attribute for every node dictionary, however some documents already contain "type" attribute which describes the category the location belongs to (e.g. Kenneth Copeland Airport's type: public, Starbucks' type: cafe). So I changed the Python scripts to take this situation into account and rename the pre-existing "type" attribute to "location_type".

- **Abbreviated Street Names**

By auditing the `addr:street` attribute, I listed and saved the most common abbreviated street types in the `mapping` list for update purpose. Some abbreviated street types are “Ln”, “Hwy”, “Expy”, “Dr”, “Pky”, “Blvd”, etc. In addition to abbreviation problems, there are misspelling and mixed-case issues, such as “Expessway” and “HIghway”.

Example:
`"S Central Expy"` will be converted to `"South Central Expressway"`.

- **Inconsistent Postal Codes**

I also find out that postal odes take different formats (“TX7xxxx”, “7xxxx-xxxx”). So I transformed any inconsistent postal codes to the 5-digit format (7xxxx).

Part 2: Overview of the Data

- **File Size**

* `dallas_texas.osm` : 636MB
* `dallas_texas.osm.json` : 679MB

- **Import Dataset to MongoDB**

In the terminal, type in the following command to import JSON data file into MongoDB:

```
> mongoimport --db OpenStreetMaps --collection dallas --type json --
file data/dallas_texas.osm.json
```

Then switched to the database I will use.

```
> use OpenStreetMaps
```

- **Statistics of Dallas Map Data**

Number of documents

```
> db.dallas.find().count()
  3022261
```

Number of nodes and ways

```
> db.dallas.find({"type":"way"}).count()
```

```
  310384
> db.dallas.find({"type":"node"}).count()
  2711877
```

Number of unique users

```
> db.dallas.distinct("created.user").length
  1808
```

Top 10 contributing users

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

Results:

```
{ "_id" : "woodpeck_fixbot", "count" : 1096185 }
{ "_id" : "Stephen Sprunk", "count" : 195229 }
{ "_id" : "fmmute", "count" : 98754 }
{ "_id" : "TexasNHD", "count" : 88350 }
{ "_id" : "25or6to4", "count" : 63918 }
{ "_id" : "Chris Lawrence", "count" : 60216 }
{ "_id" : "brianboru", "count" : 56524 }
{ "_id" : "balrog-kun", "count" : 55521 }
{ "_id" : "Dami_Tn", "count" : 55409 }
{ "_id" : "DaveHansenTiger", "count" : 44202 }
```

Top 10 mentioned cities

```
> db.dallas.aggregate([{"$match":{"address.city":{"$exists":1}}},
                       {"$group":{"_id":"$address.city",
                                  "count":{"$sum":1}}},
                       {"$sort":{"count":-1}},
                       {"$limit":10}])
```

Results:

```
{ "_id" : "Frisco", "count" : 49575 }
{ "_id" : "Plano", "count" : 3053 }
{ "_id" : "Dallas", "count" : 760 }
{ "_id" : "Cedar Hill", "count" : 628 }
{ "_id" : "Fort Worth", "count" : 437 }
{ "_id" : "Arlington", "count" : 298 }
{ "_id" : "McKinney", "count" : 293 }
```

```
{ "_id" : "Grand Prairie", "count" : 248 }
{ "_id" : "Irving", "count" : 132 }
{ "_id" : "Denton", "count" : 128 }
```

Top 10 appearing amenities

```
> db.dallas.aggregate([{"$match":{"amenity":{"$exists":1}}},
                       {"$group":{"_id":"$amenity",
                                  "count":{"$sum":1}}},
                       {"$sort":{"count":-1}},
                       {"$limit":10}])
```

Results:

```
{ "_id" : "parking", "count" : 4181 }
{ "_id" : "place_of_worship", "count" : 2856 }
{ "_id" : "school", "count" : 1934 }
{ "_id" : "fast_food", "count" : 1133 }
{ "_id" : "restaurant", "count" : 1046 }
{ "_id" : "fuel", "count" : 560 }
{ "_id" : "bank", "count" : 235 }
{ "_id" : "grave_yard", "count" : 232 }
{ "_id" : "fire_station", "count" : 166 }
{ "_id" : "post_office", "count" : 154 }
```

Top 10 appearing leisure facilities

```
> db.dallas.aggregate([{"$match":{"leisure":{"$exists":1}}},
                       {"$group":{"_id":"$leisure",
                                  "count":{"$sum":1}}},
                       {"$sort":{"count":-1}},
                       {"$limit":10}])
```

Results:

```
{ "_id" : "pitch", "count" : 2014 }
{ "_id" : "park", "count" : 1359 }
{ "_id" : "swimming_pool", "count" : 402 }
{ "_id" : "playground", "count" : 275 }
{ "_id" : "sports_centre", "count" : 94 }
{ "_id" : "track", "count" : 86 }
{ "_id" : "golf_course", "count" : 78 }
{ "_id" : "stadium", "count" : 70 }
{ "_id" : "slipway", "count" : 39 }
```

```
{ "_id" : "dog_park", "count" : 24 }
```

Top 5 appearing "cfcc" (Census Feature Class Codes)

```
> db.dallas.aggregate([{"$match":{"tiger.cfcc":{"$exists":1}}},
                       {"$group":{"_id":"$tiger.cfcc",
                                  "count":{"$sum":1}}},
                       {"$sort":{"count":-1}},
                       {"$limit":5}])
Results:
{ "_id" : "A41", "count" : 142522 }
(A41:Local, neighborhood, and rural road, city street, unseparated)
{ "_id" : "A74", "count" : 8368 }
(A74:Private road or drive for service vehicles, usually privately
owned and unnamed. Primary type of use is for access to oil rigs,
farms, or ranches)
{ "_id" : "A63", "count" : 5848 }
(A63:Access ramp, the portion of a road that forms a cloverleaf or
limited access interchange)
{ "_id" : "A45", "count" : 3746 }
(A45:Local, neighborhood, and rural road, city street, separated)
{ "_id" : "A64", "count" : 3425 }
(A64:Service drive, the road or portion of a road that provides access
to businesses, facilities, and rest areas along a limited-access
highway; this frontage road may intersect other roads and be named)
```

Most popular fast food

```
> db.dallas.aggregate([{"$match":{"amenity":{"$exists":1},
                                  "amenity":"fast_food"}},
                       {"$group":{"_id":"$cuisine",
                                  "count":{"$sum":1}}},
                       {"$sort":{"count":-1}},
                       {"$limit":1}])
Result:
{"_id": "burger", "count": 390}
```

Part 3: Additional Thoughts about the Dataset

Many "node" elements in the OSM XML dataset deliver little information.

During the data wrangling phase of this project, I find too many elements containing only latitude and longitutde data without any child element to describe more about that location.

```
<node changeset="3536198" id="82622101" lat="32.724415"
lon="-97.345584" timestamp="2010-01-04T11:37:28Z" uid="147510"
user="woodpeck_fixbot" version="2" />
<node changeset="3536198" id="82622111" lat="32.721383"
lon="-97.343951" timestamp="2010-01-04T11:37:29Z" uid="147510"
user="woodpeck_fixbot" version="2" />
<node changeset="3536198" id="82622127" lat="32.7159129"
lon="-97.3449757" timestamp="2010-01-04T11:37:30Z" uid="147510"
user="woodpeck_fixbot" version="3" />
...

<node changeset="11801377" id="1776530004" lat="32.4803845"
lon="-96.4302622" timestamp="2012-06-04T21:55:54Z" uid="672878"
user="TexasNHD" version="1" />
<node changeset="11801377" id="1776530014" lat="32.4807145"
lon="-96.4332769" timestamp="2012-06-04T21:55:54Z" uid="672878"
user="TexasNHD" version="1" />
<node changeset="11801377" id="1776530050" lat="32.481275"
lon="-96.4316171" timestamp="2012-06-04T21:55:55Z" uid="672878"
user="TexasNHD" version="1" />
```

Many of these elements are actually generated by users like "woodpeck_fixbot" and "TexasNHD". Some are robots that carry out automated or semi-automated edits maintain OpenStreetMap, some are experienced users who get consultation from the local community, and some are new contributors who have not been familiar with the whole validation process yet.

As a result, it is difficult for us to detect the reliability of these edits, so the contributors' reputation could play an important role here. The node or way elements could add attributes which could show active days of that contributor, how many edits were

confirmed by official institutes and so on. Thus, it will do us a great favor to filter out possibly incorrect edits and speed up the data wrangling process a little bit.

So how can we improve the quality of map dataset here? Use third party tools like Google Map API might be a good idea. Google map API is a reliable and popular map data source which many users trust and use around the world. If there're any ambiguous location nodes in the OSM dataset, I can use Google map API to validate that location, thanks to the advanced features like real-time traffic, routing (directions), geocoding (translating addresses into locations), 3D buildings and 'bird's eye' views, street-level imagery, and information about businesses on the map.

However, Google controls all parts of its map, thus giving less flexibility to users. API is not always free food, you have to pay for it if you reach a certain number of map uploads each day. Besides, Google lacks some geo information in remote areas, which is a big disadvantage compared to open data system like OSM.

Conclusion

This project is extremely comprehensive and intensive, covering topics like data wrangling (Python) and data analysis (MongoDB). It doesn't surprise me much that almost 80% of my time and energy were spent on the data wrangling phase. Data wrangling is cumbersome work but worth every minute as it determines the level of reliability in the final statistical results. The review of the data is cursory, though I believe the data is well cleaned for the purpose of this exercise.

References

Udacity Data Wrangling with MongoDB Course

Sample Data Wrangling Project1 by Matthew Banbury

Sample Data Wrangling Project2 by AllenReyes

OpenStreetMap Wiki Page

Detect OSM changeset with incorrect edits

MongoDB Query Documents

Python ElementTree Documents

Guide to Google map API