Wrangle OpenStreetMap Data

1 Introduction

I chose the rural district of Neustadt an der Aisch-Bad Windsheim, Germany as the basis for data wrangling. It is my home county, so I have a personal relationship with that corner of the earth. As I grew up there, I think I know the peculiarities of the region pretty well.

I used Mapzen for extracting the data. My custom extract was created on 2017 November 02, at 06:25 PM. The uncompressed file is ca. 350MB.

The bounding box was as follows:

10.0696134999998 - 10.82142999999979,
49.41005199999984 - 49.75625999999986
The URL link to the map position/area is http://www.openstreetmap.org/#map=11/49.5838/10.5654

2 Problems encountered in my map

2.1 Problems encountered in my map

This chapter is the result of the audit phase – auditing means finding the errors and finding a mapping to automatically correct it. I used a sample containing 2.5% of the whole data set to get a first impression of the problems in my OSM data set; I extracted the unique tag keys and values using a small Python program.

The first issue was the "addr:street" tag; it contained abbreviated street names like “XXX Str.” or “Xxxxstr.”, but those cases were relatively infrequent (3-4 occurrences in the whole data set). I used a regular expression to find the occurrences and replace them with the properly spelled versions (e.g. “Straße”)

Another common issue were irregularly structured phone and fax numbers. The following list gives examples for the two common problem cases that I programmatically corrected:

- Dashes between the country code and the rest of the number (+49-9302-988781).
- missing “+” sign before the country code (4998413372)

The gold standard for number consistency is the German standard https://de.wikipedia.org/wiki/DIN_5008 (c.f. https://de.wikipedia.org/wiki/Rufnummer) e.g.

- +49 30 12345-67,
- +49 30 1234567,
- 030 12345-67,
- (030) 12345 67

I used a regular expression to detect each of the two issues, and string operations to fix them. The dash removal procedure correctly treats optional “(0)” after the country code (+49-(0)123456)

Another issue is associated with the “addr:city” key. For example, the city “Höchstadt a.d.Aisch” contains “a.d.” as an abbreviation for “an der”. The data set contains variants with additional whitespace between the letters. I used a regular expression to find the issues; it can deal with additional and missing whitespace. I used the re.sub() function to replace the matching portion with the expanded version “ an der “.

Also, the "name” key had some consistency issues. The values contained “FF”, “FFW”, “Feuerwehr”, “Freiwillige Feuerwehr”, “Freiw. Feuerwehr” which all mean “voluntary fire brigade” in German. Again, a regular expression and the re.sub() function were used to replace the offending instances with the consistent name “Freiwillige Feuerwehr”.

Postcodes were inspected, but no cleaning was necessary.

The complete data set contains the following results key types:

- 'lower letters': 538431,
- 'lower letters with colon': 155907,
- 'other': 13088,
- 'problemchars': 0

It also contains the following tag types:

- 'node': 1668054,
- 'nd': 2090359,
- 'bounds': 1,
- 'member': 67843,
- 'tag': 707426,
- 'relation': 3023,
- 'way': 251862,
- 'osm': 1

2.2 Problems encountered with processing map data in Python

As the map data contained UTF-8 characters for street names etc., the Python programs needed to be UTF-8 capable. This posed the be harder than anticipated (c.f. https://pythonhosted.org/kitchen/unicode-frustrations.html) The treatment of UTF-8 strings in Python2.7 collections (set, dict etc.) was hard to work with, especially for output operations.

3 Overview of the data

After cleaning the data set, I imported the complete data set into a PostgreSQL database for further analysis using the provided Python programs. The cerberus schema checker did not detect any flaws during validation.

SQL database queries will now used to provide a statistical overview of the data set.

Size of the files:

```
nodes_tags.csv: 5.4MB
nodes.csv: 129.4 MB
ways_nodes.csv: 47.4 MB
ways_tags.csv 17.4 MB
ways.csv: 14.1 MB
NEA_District.osm: 350.8 MB
```

PostgresSQL Database size; also containing meta-data:

```
-- Size of the project database
select pg_database_size('udacity')/(1024*1024) megabytes;
 megabytes
-----------
       341
(1 Zeile)
```

Number of unique users:

```
SELECT COUNT(DISTINCT(e.uid)) FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM
ways) e;
 count
-------
  1708
(1 Zeile)
```

Number of nodes:

```
SELECT COUNT(*) FROM nodes;
  count
---------
 1668054
(1 Zeile)
```

Number of ways:

```
SELECT COUNT(*) FROM ways;
 count
--------
 251862
(1 Zeile)
```

Top10 contributing users:

```
SELECT e.username, COUNT(*) as num FROM
       (SELECT username FROM nodes UNION ALL
         SELECT username FROM ways) e
GROUP BY e.username
ORDER BY num DESC
LIMIT 10;

      username       |  num
---------------------+--------
 Rosty               | 141610
 slint               | 122745
 gsperb              | 114583
 travelling_salesman |  88238
 theophrastos        |  73485
 Kartenfreund        |  59751
 CaptainCrunch       |  56563
 geodreieck4711      |  54159
 Olikos              |  53839
 BikingTom           |  51853
(10 Zeilen)
```

Number of users appearing only once (having 1 post)

```
SELECT COUNT(*) FROM
       (SELECT e.username, COUNT(*) as num FROM
               (SELECT username FROM nodes UNION ALL SELECT username FROM ways) e
       GROUP BY e.username HAVING COUNT(*)=1) u;
 count
-------
   277
(1 Zeile)
```

As the district is a pretty rural place, hunting stands are one of the most documented amenities in the data set. Let’s see how those hunting stands are additionally tagged :

```
select  key,value from (SELECT id FROM nodes_tags WHERE key='amenity' and
value='hunting_stand') v_hs inner join nodes_tags t2 on t2.id=v_hs.id
where not t2.key='amenity' and not t2.value='hunting_stand' and not key='fixme' and
not key='ref' and not key='source' and not t2.key='hunting_stand'
group by key, value order by key ;
    key    |             value
-----------+------------------------------
 covered   | yes
 de        | Birke
 height    | 3
 height    | 5
 height    | full
 height    | half
 height    | low
 hide      | no
 hide      | yes
 ladder    | yes
 leaf_type | broadleaved
 lockable  | no
 lockable  | yes
 man_made  | tower
 name      | SV Hubertus Neuhaus 1952 e.V.
 natural   | tree
 raised    | yes
 shelter   | no
 shelter   | yes
 type      | communication
```

So it seems we have a wide variety of hunting stands.

4 Other ideas about the data set

4.1 Additional improvements

There are a number of possible improvements that have not been addressed in the overview above, but could be beneficially implemented.

There are many tags that feature URL addresses, e.g.

```
<tag k="website" v="http://www.vgn.de/komfortauskunft/ttb/?lineName=714"/>,
<tag k="url" v="http://www.adelshofen.de"/>
```

Those addresses could be checked for broken links; you could write a small Python program, that queries the given URL address e.g. by using the Python package BeautifulSoup (https://www.crummy.com/software/BeautifulSoup/bs4/doc/). If the query returns an HTML error or a network timeout, the address is probably no longer valid and should be manually inspected (and maybe replace by a working one).

In a similar vein, some organizations also include their email address as tags in the OSM data set, e.g.

```
<tag k="email" v="info@hotel-rappen-rothenburg.com"/>
```

You could check the well-formedness of these email addresses, e.g. by checking if the address contains an @-character. Here is a sample query for this simple case:

```
SELECT value from
(select key, value  from nodes_tags union all select key, value from ways_tags)
v_tags where
key='email' and
not value ~* '.+@.+';

                  value
------------------------------------------
 haus-der-senioren[at]awo-unterfranken.de
(1 Zeile)
```

Of course, more sophisticated checks are possible, although it will be hard to detect malformed email addresses by regular expression alone. You would probably need a directory of valid top-level domains as a gold standard to check the accuracy of the TLD part of the address. Or you could do WHOIS lookups to determine the status of the server address.

Some tag keys contain the placeholder string "FIXME" or "TODO". Those elements should also be inspected and modified with a more meaningful content. These keys are good starting points for supporters who would like to contribute to OSM.

The following query identified these elements:

```
select key, count(key) from
(select key from nodes_tags union all select key from ways_tags) v_tags
where key like 'FIXME' or key like 'fixme' or key like 'TODO'  group by key;

   key  | count
-------+-------
 fixme |  1900
 FIXME |   880
 TODO  |   148
(3 Zeilen)
```

I checked the whole list of tag keys, but the spelling of the TODO key was unique.

If you check all tag keys which contain the phrase "phone", it becomes clear that there are different types of phone numbers documented in the OSM data set. Here is the complete list of the keys:

phone	1341
contact:phone	240
openGeoDB:telephone_area_code	111
communication:mobile_phone	18
payment:telephone_cards	6
phone:mobile	6
emergency_telephone_code	4
phone_1	3
telephone	3
authentication:phone_call	2
phone2	1
contact:phone2	1

Two other ways to improve German phone and fax numbers are listed here:

- Remove slashes, which separate the prefix code from the number (e.g. 09103/8436). This spelling is a common one, but it is not permissible according to the above mentioned standard.
- Remove brackets around numbers, that must be left out when using the +49 country code (e.g. +49 (0)9331 3171). The "(0)" can be removed.

4.2 Discussion of benefits and problems

Checking the validity of URL and email addresses would be a big quality improvement, as these addresses have become more and more relevant in the last years, and they tend to become invalid without notice. The problem with fixing these addresses lies in the manual effort required for this task. It is hard to automatically check the validity of an email address; the only sure way would be to actually send a request email to the address and assess the response.

URL address may be easier to check automatically, but eventually, a human judge is needed to confirm that the Web site is indeed meaningful and consistent with the URL. (E.g. a domain could still be reachable, but the Web content could be meaningless.)

Improving the consistency of phone and fax number tag keys seems to be a good candidate for automation (e.g. combining “phone” and “telephone”), but there may be semantic differences in the keys – is a “contact:phone” the same as a “phone”? According to OSM guidelines, users are pretty free to name their tags as they see fit. So an automatic correction can only target a small subset of categories.

The “FIXME” or “TODO” keys obviously need improvement, but the values associated with these keys are in natural language, hence it is very hard to understand and make sense of for a computer, and the improvement would require intimate knowledge of the location at hand. So, an automated improvement seems unlikely.

5 Conclusion

After a holistic look at the data, it is remarkably clean and orderly. There are a few quality problems here and there, and some of them can be cleaned automatically for the purpose of this exercise. As a next step, the corrected elements should be imported back into the OSM, after discussing the changes with the OSM community and the original contributors.