OpenStreetMap

Data Wrangling with MongoDB

Stefan Borchardt

Map Area: Berlin, Germany

1. Problems Encountered in the Map

After the download of the preselected area of Berlin, Germany from MapZen I encountered problems parsing the file iteratively because of insufficient memory. I chose to use the SAX parser instead of ElementTree, which was used in the lesson. This way the memory consumption is lower, but the code complexity increases.

The file size was still too large for implementing the SAX ContentHandler or for exploring which problems in the data can be handled programmatically before the import into MongoDB. So I implemented a way to slice a chunk from the original data file for development purposes.

After these preliminary steps I encountered the following problems in the map data. Numbers given in this section were obtained through counters during processing unless stated otherwise.

Problematic characters in `k` attributes of `tag` elements

Dots '.' were the most common source of keys with problematic characters. Given that '.' and ':' are on the same key of the German keyboard layout, that is not surprising. But '+', '-' and even whitespace characters were also found. Together, 105 problematic keys were found and omitted in a total of 6,984,341 tag elements.

Elements Tagged with 'fixme'

Some `node` and `way` elements had `tags` with a key of *'fixme'* or *'FIXME'*, the value of which explained doubts about correctness or completeness of the element. These 2,289 top-level elements were skipped (of 10,824,008 top-level elements). Some of the skipped elements that I looked into seem to be in Poland, which can be explained by the size of the preselected area, because I made similar findings with the postcodes.

Implausible Postcodes

Many postcodes were tagged to elements as '*postal_code*' and have been moved to the address dictionary. Some postcodes did not have 5 digits, as they should have in Germany, or a whole list was given as value. These cases were omitted. Almost all of these omitted postcodes (9,915 to be precise) were in a format that seems to be Polish, compared to 600,544 proper postcodes. A later check in MongoDB revealed German postcodes that are approximately 100 miles from Berlin:

```
[{'$match': {'address.postcode': {'$exists': True}}},
 {'$group': {'_id': '$address.postcode', 'count': {'$sum': 1}}},
 {'$sort': {'count': 1}}]
```

The most frequent postcodes are from the surrounding state of Brandenburg, nevertheless the vast majority seems to be from Berlin and the wider metropolitan area. Maybe MapZen is a bit generous with the preselected area.

Abbreviated Street Names

In terms of street names, the situation is complicated in Germany. There is an interesting discussion on the issue in http://forum.openstreetmap.org/viewtopic.php?id=19313 (in German). In short, there are just too many correct ways for where and how the street type can be put in the street name. I checked for the most common abbreviation and misspelling, and 17 cases were found, while 560,286 were inconspicuous. The bot seems to do a good job there.

Variety of Tags

I found a total of 3325 unique, importable keys in the map data. To be more specific, of the 32 unique tags starting with 'addr:' (without a second ':', as specified) less than a quarter are actually used frequently and half of the tags is used less than 10 times (among approximately 500,000 addresses). A white list could be helpful, but is not intended by the specification.

For the non-address keys, the white-list approach is more difficult to implement without further information about in which direction the data is to be investigated. The 300 most frequent keys are a wild mixture from detailed like '*roof:orientation*' over practical like '*shop*' or '*amenity*' to unclear like '*adr_les*' (used 1616 times), which http://wiki.openstreetmap.org/wiki/Key:adr_les explains as '*???*'. To make use of the information buried in these tags a lot of omitting and merging might be necessary.

2. Data Overview

This section contains basic statistics about the dataset and the MongoDB queries used to gather them.

File sizes

```
berlin_germany.osm ......... 2.1 GB
berlin_germany.json ........ 2.3 GB
```

I had not expected the JSON to be larger than the XML, but OSM XML uses mainly empty elements and places the content in the attributes, so this is plausible. The number of top-level elements processed equals the number of documents found in the database:

```
# Number of documents
      db_collection.find().count()
      10,821,720

# Number of nodes
      db_collection.find({"type":"node"}).count()
      9,395,813

# Number of ways
      db_collection.find({"type":"way"}).count()
      1,423,915


# Number of unique users
      len(db_collection.distinct("created.user"))
      7,056
```

Alphabetical list of amenities tagged more than 250 times

```
[{'$match': {'amenity': {'$exists': True}}},
{'$group': {'_id': '$amenity', 'count': {'$sum': 1}}},
{'$match': {'count': {'$gt': 250}}},
{'$project': {'_id': True}},
{'$sort': {'_id': 1}}]
```

```
[{u'_id': u'atm'},
 {u'_id': u'bank'},
 {u'_id': u'bar'},
 {u'_id': u'bench'},
 {u'_id': u'bicycle_parking'},
 {u'_id': u'cafe'},
 {u'_id': u'car_wash'},
 {u'_id': u'dentist'},
 {u'_id': u'doctors'},
 {u'_id': u'fast_food'},
 {u'_id': u'fire_station'},
 {u'_id': u'fountain'},
 {u'_id': u'fuel'},
 {u'_id': u'grave_yard'},
 {u'_id': u'hunting_stand'},
 {u'_id': u'kindergarten'},
 {u'_id': u'parking'},
 {u'_id': u'parking_space'},
 {u'_id': u'pharmacy'},
 {u'_id': u'place_of_worship'},
 {u'_id': u'post_box'},
 {u'_id': u'post_office'},
 {u'_id': u'pub'},
 {u'_id': u'public_building'},
 {u'_id': u'recycling'},
 {u'_id': u'restaurant'},
 {u'_id': u'school'},
 {u'_id': u'shelter'},
 {u'_id': u'social_facility'},
 {u'_id': u'taxi'},
 {u'_id': u'telephone'},
 {u'_id': u'toilets'},
 {u'_id': u'vending_machine'},
 {u'_id': u'waste_basket'}]
```

3. Additional Ideas

Data exploration with MongoDB: What amenities are near the police?

To answer this question a geospatial index on the '*pos*' field had to be added to the collection. Because MongoDB's `$geoNear` aggregation pipeline stage takes only one location at a time and has to be first in the pipeline, I first ran a query to obtain all locations of the police. Then I queried separately for the amenities 1000m near each of the locations and merged the counts using a Python counter.

Here is the pipeline for obtaining the amenities near the police:

```
[{'$geoNear': {'near': {'type': 'Point', 'coordinates': coord},
               'distanceField': "distance",
               'maxDistance': 1000,
               'spherical': True}},
 {'$match': {'$and': [{'amenity': {'$exists': True}},
                      {'distance': {'$gt': 0}}]}}]
```

On the left the top 10 amenities near the police, on the right the overall top 10:

```
[(u'restaurant', 19),          [{u'_id': u'parking', u'count': 12771},
 (u'parking', 18),              {u'_id': u'bench', u'count': 7208},
 (u'bench', 17),                {u'_id': u'restaurant', u'count': 5352},
 (u'waste_basket', 14),         {u'_id': u'post_box', u'count': 3422},
 (u'post_box', 12),             {u'_id': u'recycling', u'count': 3091},
 (u'fast_food', 10),            {u'_id': u'kindergarten', u'count': 2781},
 (u'police', 10),               {u'_id': u'fast_food', u'count': 2544},
 (u'toilets', 8),               {u'_id': u'bicycle_parking', u'count': 2489},
 (u'cafe', 8),                  {u'_id': u'cafe', u'count': 2354},
 (u'bicycle_parking', 8)]       {u'_id': u'school', u'count': 2022}]
```

It seems that restaurants and waste baskets are found more frequently near the police, in this dataset, but that would have to be investigated further.

Suggestion for improving data quality

There seems to be an active community of map enthusiast, who use tools like http://wiki.openstreetmap.org/wiki/Osmfilter or http://taginfo.openstreetmap.org/ to investigate and improve the data quality programmatically, as I saw when handling the street types. Nevertheless, I found tags that are marked as deprecated in the OSM wiki or were misspellings of http://wiki.openstreetmap.org/wiki/Map_Features. Continuing and extending programmatic data cleaning is probably the best way to improve data quality, but that also requires a lot of expertise in the domain.

Conclusion

I was amazed by the amount and detail of the data available on OpenStreetMap. The quality, in terms of consistency, accuracy and also completeness still has potential, though. Knowing in which direction the data exploration is going to be would help when wrangling the data.