Project 3: Data Wrangling OpenStreetMap with MongoDB

Submitted by Rebecca Mayer, September 22, 2015
https://github.com/rbmayer/Project3.git

Contents

Section 1. Map Audit

Background

For this project I extracted OpenStreetMap (OSM) data on Syria using the Overpass API. After exploring the map online and opening a small subset of the XML source data in Google Chrome, I made a few observations:

- geographic data was fairly clean and sparse, with major roadways and buildings labelled but very few street addresses
- attribute names were listed in multiple languages, usually arabic and english
- most data was added in the past several years with updates continuing in 2015.

A glance through the OSM wiki showed that the Humanitarian OpenStreetMap Team (HOT) organized an update of priority areas in Syria about two years ago. This may explain the relatively clean and uniform quality of the data.

To come up with a plan for this assignment, I imagined a use case in which relief organizations needed to complete the labelling and translation of all street names into both english and arabic in order to coordinate logistics with english-speaking and arabic-speaking partners. I created a plan to use mongoDB to conduct an inventory of street name languages and to identify nodes for translation by crowdsourced volunteers.

Section 2. Problems Encountered in Data Cleaning and Processing

The following steps comprised the data auditing, cleaning and shaping process and associated code files:

1. Count tag types (mapparser.py)
2. Check for problem characters that can't be used as keys in mongoDB (tags.py)
3. Count number and type of tag attributes containing “addr” to quantify scope of potential cleaning (audit_find_addr.py)
4. Check street names for abbreviations and errors in english and arabic (audit_street_names.py)
5. Clean issues raised in #4, add language tags to street names, and output shaped data to JSON (project3.py)
6. Import JSON data into mongoDB (dbinsert.py)
7. Run queries on data (queries.py)

Unicode processing

The biggest challenge, by far, was dealing with non-ascii unicode characters in the pre-processing of the XML data into JSON. When using cElementTree iterparse(), there were many errors of the type "UnicodeDecodeError: 'ascii' codec can't decode…". This happened despite the encoding setting on the first line of the script (# -*- coding: utf-8 -*-) and whether or not I called .encode('utf-8') or .decode() on specific strings.

Ned Batchelder's youtube presentation "*Pragmatic Unicode, or, How do I stop the pain?* " was extremely helpful in clarifying the difference between unicode and bytes, and explaining the types of implicit conversions carried out by Python 2 that could cause the unicode-related errors noted above. Based on Batchelder's tips and other resources noted in the References, I created a number of special functions to handle parsing of arabic and english unicode text:

- **make_pairs_unicode()** checks if input values are unicode type and converts them if not
- **is_arabic()** and **is_english()** identify language using re pattern matching against the unicode code point ranges of arabic and english, respectively.
- **match_arabic_name()** extracts arabic-language street types and strips the 'RIGHT-TO-LEFT EMBEDDING' (U+202B) character, to ensure equality comparison with terms in the list of expected street types. The presence of the embedding character is only visible when the unicode strings are expressed as code points, so this was a tricky issue to debug.

The street name audit found two abbreviations and one typographical error to correct in the mapping function:

mapping = { u"شارعشارع"u : "", u"St." : u"Street", u"Rd" : u"Road" }

For each way containing a street name in english or arabic, the following tags were added to the JSON output, as appropriate, in project3.py:

{ english_street_name : 1 }

{ arabic_street_name : 1 }

These tags simplify aggregation queries in mongoDB over street names by language.

Memory constraints

Another significant problem was the progressively increasing use of memory resources as iterparse() traversed the 100 MB+ XML data file. This caused my machine to freeze the first few times I tried to parse the entire file. Review of class notes as well as Liza Daly's blog post "High-

performance XML parsing in Python with lxml" showed the need to clear unneeded elements and the root node at the end of each iterparse event. Additionally, memory during processing was reduced by dropping the python 'data' variable to which the JSON output was saved and instead saving only to an external file.

Section 3. Data Overview & Query Results

File sizes
map …….. 117.3 MB
map.json …….. 153.4 MB

Total number of streets[1]: 1008
```
db.osm_syria.aggregate({ "$match" : { "street_names" : { "$exists" : 1 } } },
        { "$group" : { "_id" : q1, "count" : { "$sum" : 1 } } } )
```

Total number of streets with arabic names only: 756
```
db.osm_syria.aggregate({ "$match" : { "street_names.arabic_street_name" : 1,
"street_names.english_street_name" : { "$exists" : 0 } } },
        { "$group" : { "_id" : q2, "count" : { "$sum" : 1 } } })
```

Total number of streets with english names only: 193
```
db.osm_syria.aggregate({ "$match" : { "street_names.english_street_name" : 1,
"street_names.arabic_street_name" : { "$exists" : 0 } } }, { "$group" : { "_id" : q3, "count" : {
"$sum" : 1 } } })
```

Total number of streets with both arabic and english names: 56
```
db.osm_syria.aggregate({ "$match" : { "street_names.english_street_name" : 1,
"street_names.arabic_street_name" : 1 } }, { "$group" : { "_id" : q4, "count" : { "$sum" : 1 } } })
```

Total number of streets with names in other languages: 3
```
db.osm_syria.aggregate({ "$match" : { "street_names" : { "$exists" : 1 },
"street_names.english_street_name" : { "$exists" : 0 }, "street_names.arabic_street_name" : {
"$exists" : 0 } } }, { "$group" : { "_id" : q5, "count" : { "$sum" : 1 } } })
```

Total number of documents: 611,226
```
db.osm_syria.aggregate({ "$group" : { "_id" : q1, "count" : { "$sum" : 1 } } })
```

Number of documents timestamped since the beginning of the civil war on March 15, 2011: 590,134

[1] During the processing stage, I defined 'streets' as any of 15 highway types in the OSM data definitions whose descriptions seemed to correspond to the concept of a street.

db.osm_syria.aggregate({ "$match" : { "created.timestamp" : { "$gt" : "2011-03-11"} } }, { "$group" : { "_id" : q3, "count" : { "$sum" : 1 } } })

Number of nodes within 5 km of Palmyra: 4656
db.osm_syria.find({"geometry" : { "$near" : { "$geometry" : { "type" : "Point", "coordinates" : [38.2809746, 34.5560155]}, "$maxDistance" : 5000 }}})

Section 4. Additional Ideas

The results of these queries could be used to set up tasks in the HOT tracker for volunteer translators and map-builders to implement. This would be beneficial by facilitating the crowdsourcing effort to improve the data by automatically quantifying the tasks to be done, as well as tracking progress over time.

It might also benefit humanitarian agencies if they could prioritize street name translations by city or region. To do this, one could focus the queries on certain geographic areas using the $near operator and mongoDB's geospatial indexing functionality.

I revised the element shaping function in project3.py to output the latitude and longitude coordinates in GeoJSON format as follows:

```
{ u'geometry' :  {"type" : "Point", "coordinates" : [lon, lat] }
```

Then I created a 2dsphere index in dbinsert.py:

```
db.osm_syria.create_index([("geometry", pymongo.GEOSPHERE)])
```

However, I soon discovered that OSM way points, which contain almost all of the street name data in this dataset, do not explicitly contain spatial coordinates. Thus they could not be queried with $near or $geoNear. Way points are composed of nodes, and the nodes have spatial coordinates. In order to run queries on street names in different localities, one would need to cross-reference the ways with the coordinates of their constituent nodes. Since streets by definition extend over broad areas, there might also be some complications in determining which part of a street qualifies as "near" to the target coordinates. I have left this as a future exercise.

Section 5. References

Batchelder, N. (2012, March). *Pragmatic Unicode, or, How do I stop the pain?* Retrieved from https://www.youtube.com/watch?v=sgHbC6udIqc&feature=results_video&playnext=1&list=PL64CA5D679C54E4E9

Daly, Liza. High-performance XML parsing in Python with lxml. (2011, March 24). [CT316]. Retrieved September 18, 2015, from http://www.ibm.com/developerworks/xml/library/x-hiperfparse/

ʞɔıu. (n.d.). regex - python and regular expression with unicode - Stack Overflow. Retrieved September 15, 2015, from http://stackoverflow.com/questions/393843/python-and-regular-expression-with-unicode

python - How to write UTF-8 in a CSV file - Stack Overflow. (n.d.). Retrieved September 16, 2015, from http://stackoverflow.com/questions/18766955/how-to-write-utf-8-in-a-csv-file

The ElementTree iterparse Function. (n.d.). Retrieved September 18, 2015, from http://effbot.org/zone/element-iterparse.htm