Open Street Map Data Wrangling with MongoDB

John Baker

Open Street Map Data Selection and Key Categorization

For this project analysis I chose a familiar map region encompassing Brevard County, FL. The custom region bounds are:

	Minimum	Maximum
Latitude	27.7224	28.7036
Longitude	-81.0126	-80.4166

I used the Overpass API link to download the data in OSM XML format.

Brevardcty.osm – 97MB

The OSM format uses free form tag key-value pairs to describe features of each OSM primitive: node, relation, and way. Keys are generally established by convention and the meaning and value sets can be found through taginfo, but users are free to create their own tags from a Unicode string. Values can follow OSM convention, but are typically free form names, descriptions and so forth.

The first step in the audit was to categorize the keys similar to the taginfo report – Characters in keys

Category	**Unique Keys**
Plain - follow common naming	212
Colon– Plain Keys with colons inside	145
Upper/Numeric –keys with an upper case or numeric character	68
White Space – keys with a space or tab	1
Possibly Problematic – keys that include punctuation	0
Other – rest of the keys not categorized	0
TOTAL	426

Unique key value pairs for each primitive were exported to a CSV and brought into excel for review. Many name values were in Unicode, so the file had to be imported using UTF-8 format. E.g.

category	**primitive**	**tag**	**value**
colon	relation	name:ko	플로리다

This initial key – value file was used to develop the data audit and cleanup plan.

Data Audit and Cleanup

My primary focus was on correcting address data.   Address for ways are in the “name” key. Addresses for nodes are in keys with an “addr:” prefix. Multiple addresses are supported in both cases (e.g., name_1, alt_name, addr:street_1, etc.).

Street addresses can also be found in node names, for example in a bus stop node, but the format was not consistent enough to easily correct programmatically (e.g., “Robert J. Conlan Blvd/ S Kirby Ave”) – these were left as is.

In addition, some ways have address data from the TIGER database, which I used to cross-validate the name key value.

The OSM data source file was iteratively processed through an audit program:

1. Parse address values into components: optional direction prefix, street name, street type and optional direction suffix
2. Create a proposed corrected address using a dictionary to provide a mapping for each component.
3. Cross validate way data with TIGER and output inconsistencies for review.
4. Output data to CSV for review in excel.

With each pass, I added to the mapping dictionaries to get consistent corrected address values.

Issues noted with the dataset:

1. Street types and direction prefix/suffix are inconsistently abbreviated
2. Typo’s in general. Eg., ‘Road’ misspelled ‘Raod’, a street type entered as ‘Kn’instead of ‘Ln’
3. state value abbreviated or spelled out
4. postal code included State
5. Some way values look like they should be nodes - e.g., “Rockledge Police Department” is a building with a street address, but it is entered as a way primitive. I assume this is done to allow reference to other nodes without going through a relation.

The cross validation with TIGER pointed out a number of inconsistencies in the data:

1. The base name for ways didn’t match (e.g, Lake View vs Lakeview, St Marks vs Saint Marks, Bee Line vs Beachline, Bluff vs Crecent)
2. Street types didn’t match (e.g., Dr vs Lane)

The TIGER cross validation was useful for identifying potential issues with names, but the problems would require additional research to confirm.   I left that out of scope for this project.

For clean up, I moved the audit logic into a separate module for exporting corrected data in JSON format.   JSON documents have the following top level keys:

Key	**Value**
primitive	('node' | 'relation' | 'way')

attrib	a dictionary of key value pairs of the primitive’s attributes
tag	dictionary of tag key value pairs
nd	list of nd ref values
member	list of dictionary of member key value pairs

Keys with an “addr:” prefix are parsed into a dictionary with a key for each part

Longitude and latitude values are stored in GeoJSON format to allow for mongoDB ‘2dsphere’ indexing.

JSON data was imported into a local instance of mongoDB into the collection “brevardcty”.

Brevardcty.json – 123MB

Overview Data Queries

The total number of documents by primitive type is determined by an aggregate query on the collection using this pipeline (*note: shown pipelines are for pymongo*):

```
[{'$group': {'_id': '$primitive', 'total': {'$sum': 1}}}].
```

Result: `{'_id': 'relation', 'total': 520} {'_id': 'way', 'total': 45205} {'_id': 'node', 'total': 448122}`

Instead of using the mongoDb ‘distinct’ collection method, the total number of unique users is determined by creating a set of user names, unwinding the set and then counting the result:

```
[{'$group': {'_id': None, 'userset': {'$addToSet':
'$attrib.user'}}}, {'$unwind': '$userset'}, {'$group':
{'_id': None, 'count': {'$sum': 1}}}]
```

Result: `{'_id': None, 'count': 341}`

The users with the most updates:

```
[{"$group": {"_id": "$attrib.user", "count": {"$sum": 1}}},
{"$sort": {"count": -1}}, {"$limit": 5}]
```

Result: `{'_id': 'NE2', 'count': 156605} {'_id': 'Panther37', 'count': 71204} {'_id': 'MichaelG68', 'count': 59939} {'_id': 'ingalls', 'count': 46018} {'_id': 'grouper', 'count': 33274}`

The top three amenity values are found by selecting all documents with a ‘tag.amenity’, grouping the results to get a count, sorting descending and returning the top three results.

```
[{'$match': {'tag.amenity': {'$ne': None}}}, {'$group':
{'_id': '$tag.amenity', 'total': {'$sum': 1}}}, {'$sort':
{'total': -1}}, {'$limit': 3}]
```

Result: `{'_id': 'parking', 'total': 1171} {'_id': 'school', 'total': 290} {'_id': 'place_of_worship', 'total': 283}`

The total number of ways that allow horses:

```
     [{"$match": {"tag.horse": "yes", "primitive": "way"}},
           {"$group": {"_id": None, "total": {"$sum": 1}}}]
```

Result: `{'_id': None, 'total': 7}`

Geospatial Data Queries

Node location data was indexed with ‘2dsphere’ index types to allow queries using the ‘$near’ operator.

The coordinates for “Da Kine Diego’s Insane Burrito” is found with a simple find query including a projection to return the coordinates:

Query - `{"tag.name": "Da Kine Diego's Insane Burrito"}`

Projection - `{"_id": 0, "attrib.pos.coordinates": 1}`

Result: `{'attrib': {'pos': {'coordinates': [-80.5902917, 28.17372]}}}`

The list of bus stops within 500 meters of Da Kine Diego’s is found using a query with the $near operator:

Query - `{"tag.highway": "bus_stop", "attrib.pos": {"$near": {"$geometry": {"type": "Point", coordinates": [-80.5902917, 28.17372]}, "$minDistance": 0, "$maxDistance": 500}}}`

Projection - `{"_id": 0, "tag.name": 1}`

Result –

```
{'tag': {'name': 'N Highway A1a/ Cassia Blvd (SW Corner)'}}

{'tag': {'name': 'N Highway A1a/ Cassia Blvd (SE Corner)'}}

{'tag': {'name': 'N Highway A1a/ Park Avw (SW Corner)'}}

{'tag': {'name': 'N Highway A1a/ Park Ave (NE Corner)'}}
```

Additional Use of Data and Improvements

The OSM website points to couple of interesting python libraries that could be used to explore the data as intended.

Pyroutelib – is a simple router that can be used to provide a path between two locations on a map. This is designed to work with OSM files directly, but it could be integrated with pymongo and the document model I created for Brevard County.

Mapnik – an open source toolkit for rendering maps and it includes a python binding. This could be used to display query results and paths produced by pyroutelib.

The cross validation with TIGER data could be improved by changing some of the parsing logic to account for cases where direction and/or type is included in the base name. Total mismatches would still need to be investigated off line.

Overall the data clean-up went well for the scope I covered, but I know a couple of typos still made it through. The JSON data creation program could have a generalized spelling correction function added.

References:

"OSM XML." - *OpenStreetMap Wiki*. N.p., n.d. Web. 01 June 2015. <http://wiki.openstreetmap.org/wiki/OSM_XML>.

"Agility, Scalability, Performance. Pick Three." *MongoDB*. N.p., n.d. Web. 01 June 2015. <http://www.mongodb.org/>.