Audit & Analysis of OpenStreetMaps Data

Montreal, Quebec, Canada
Steven Newton (newton36@gmail.com)

Data from OpenStreetMaps database, downloaded from Mapzen.

Contents

1. Auditing Challenges
   - Bilingual dataset
   - Large file size
2. Data Analysis with MongoDB
3. Additional Thoughts

1. Auditing Challenges

Bilingual dataset

I began the audit with a sample .osm file from the island of Montreal imported directly from OpenStreetMaps. The street name field varied widely in formatting styles, primarily because French convention lists the street identifier first and in lowercase, e.g. “rue Dorval.” For the sake of consistency, I decided to capitalize the first word and spell out any abbreviations. This meant extending the mapping dictionary from the original *audit.py* file to include both French and English. One difficulty here is that North, South, and East all have the same first letter in both languages. I did not attempt to spell out the any occurrences of “N”, “S”, or “E” while updating the street names.

When determining whether or not a street should be updated, the function had to check the first word against a list of expected French words and the last word against a list of expected English words. If neither the first or last word was in the expected lists, the street name was updated using the bilingual mapping dictionary:

```
def audit_street(street_name):
        m = street_type_re.search(street_name) # matches last word in street name
        n = street_type_reFr.search(street_name) # matches first word in street name
        if m and m.group() not in expected and n and n.group() not in expectedFr:
                update_street_name(street_name)
                # updates street if first and last words are not in the expected values list
        else:
                return street_name
```

Another French naming custom is to include dashes between words e.g. “Sainte-Catherine.” Handling dashed words individually requires a more complex regular expression than used here.

Large file size

Moving on to the 695mb .osm file for all of Montreal, I ran a count of the tag names with these the results:

{'bounds': 1, 'member': 16572, 'nd': 3385303, 'node': 2959105, 'osm': 1, 'relation': 2202,
'tag': 2813417, 'way': 399294}

Converting this XML file to JSON took over nine hours with the original *data.py* file with my 1.8GHz AMD A4 processor. The output file was 769mb without using pretty-print. The conversion introduced unicode code points into any string with special characters, e.g. "Montréal" became "Montr\u00e9al". This was not my intention. Early attempts to preserve the original formatting resulted in *UnicodeDecodeError* and "*character maps to <undefined>*" messages. I learned that the Python 2.7 version I was using treats <'str'> objects as a string unless explicitly prefixed with 'u' for unicode. This explained the first error. The mapping error required adding *encoding='utf-8'* as a parameter when opening the output file and *ensure_ascii=False* when writing.

After the first attempt to convert the .osm file, I discovered (from Stack Overflow forums) two Python modules implemented in C that promised to optimize my script speed: 1) cElementTree in place of ElementTree to parse the XML and 2) the io module in place of codecs to access the file (io is a default module in Python 3.x). I also added the *element.clear()* command between element loops on a recommendation in the Udacity forums. The second conversion from XML to JSON took just 98 minutes with the addition of the street update function discussed in the previous section. I was thoroughly impressed. Even with the added step of cleaning the data, the performance increase was dramatic.

Importing the JSON file into MongoDB with mongoexport.exe running in Powershell took 38 minutes and created a 1.953Gb database. In the next section, I provide the queries I ran from the mongo.exe shell, as well as the outputs and brief analyses.

2. Data Analysis with MongoDB

File sizes:
montreal.osm.....695mb
montreal.osm.json...769mb

```
> db.osm.find().count()
3358399   # documents in database

> db.osm.find({"type":"node"}).count()
2959001   # node elements in database

> db.osm.find({"type":"way"}).count()
399064   # way elements in database
```

Comment: Interestingly, the node and way counts were a fraction of a percent less than the counts from

the XML file. I am curious to know what causes the difference.

```
> db.osm.distinct("created.user").length
1593   # distinct contributing users

> db.osm.aggregate([{$project: {"address.street": 1 }}, {"$group": {"_id": "$address.street"}}])
{ "_id" : "Fielding Avenue" }
{ "_id" : "Route Marie-Victorin" }   # the updated addr:street field
...

> db.osm.aggregate([{$group: {_id: "$created.user", count: {$sum: 1}}}, {$sort: {count: -1}}, {$limit:
2}])
{ "_id" : "canvec_fsteggink", "count" : 634746 }
{ "_id" : "jfd553", "count" : 584662 }   # top two contributing users

> db.osm.aggregate([{$group: {_id: "$created.user", count: {$sum: 1}}}, {$group: {_id: "$count",
num_users: {$sum: 1}}}, {$sort: {_id: 1}}, {$limit: 1}])
{ "_id" : 1, "num_users" : 283 }   # number of users with only 1 post, _id is post count

> db.osm.aggregate([{$match: {"address.city": {$exists:1}}}, {$group: {"_id": "$address.city", count:
{$sum: 1}}}, {$sort: {count: -1}}, {$limit: 5}])
{ "_id" : "Montréal", "count" : 81862 }   # top 5 entries for city
{ "_id" : "Laval", "count" : 35782 }
{ "_id" : "Terrebonne", "count" : 11355 }
{ "_id" : "Saint-Jean-sur-Richelieu", "count" : 10580 }
{ "_id" : "Repentigny", "count" : 8360 }

> db.osm.aggregate([{$match:{"address.suburb":{$exists:1}}}, {$group: {"_id":"$address.suburb",
count:{$sum:1}}}, {$sort: {"count": -1}} , {$limit: 1}])
{ "_id" : "Le Plateau-Mont-Royal", "count" : 97 }   # most common suburb

> db.osm.distinct('address.city').length
304   # distinct cities
```

Comment: The city field contained 304 distinct user entries. While a few of these cases were the result of typos or accidental inclusion of the province, the fact remains that 'city' is an ambiguous term in Montreal.

```
> db.osm.distinct('address.postcode').length
1660
```

Comment: There are 1660 unique postcodes in the database. The most common typo is exclusion of the space mandated by Canada Post. For example the code A1A 1A1 is entered as A1A1A1 or A1A-1A1. A bit of useful trivia: the letters D, F, I, O, Q and U should never appear in a Canadian postal code.

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

{ "_id" : "parking", "count" : 2141 } *# most common amenity*
{ "_id" : "school", "count" : 1738 }
{ "_id" : "restaurant", "count" : 1206 }
{ "_id" : "place_of_worship", "count" : 1054 }
{ "_id" : "fast_food", "count" : 484 }
{ "_id" : "cafe", "count" : 406 }
{ "_id" : "fuel", "count" : 392 }
{ "_id" : "bicycle_parking", "count" : 389 }
...
{ "_id" : "bicycle_rental", "count" : 101 }
...
{ "_id" : "taxi", "count" : 29 }
...
{ "_id" : "shower", "count" : 1 } *# least common amenity (just 1 shower!)*

Comment: Bicycle parking and rental spots are prevalent in the data. Cycling is obviously a very important mode of transportation in Montreal. There are surprisingly few taxi locations in comparison.

> db.osm.aggregate([{$match: {"highway": {$exists: 1}, "highway": "cycleway"}}, {$group: {"_id": "$name", count: {$sum: 1}}}, {$sort: {count: -1}}, {$limit: 2}])
{ "_id" : null, "count" : 1460 }
{ "_id" : "route verte 1", "count" : 41 }

Comment: Quebec has an extensive bikepath system to support all that traffic called Route Verte 1. Those 1460 null entries probably include a lot of nodes on Route Verte 1.

> db.osm.aggregate([{$match: {"amenity": {$exists: 1}, "amenity": "cafe"}}, {$group: {"_id": "$name", count: {$sum: 1}}}
, {$sort: {count: -1}}])
{ "_id" : "Tim Hortons", "count" : 85 }
{ "_id" : "Second Cup", "count" : 30 }
{ "_id" : "Starbucks", "count" : 18 }
...
{ "_id" : "Dunkin Donuts", "count" : 2 }

Comment: Montrealers display their exquisite taste: the unparalleled Tim Horton's far outpaces Starbucks in number of locations. Dunkin Donuts barely has a market share.

3. Additional Thoughts

The top contributing user was "*canvec_fsteggink*". A quick Google search revealed that CanVec is a coast-to-coast spatial dataset derived from the National Topographic Database. The Department of Natural Resources has produced an .osm version of CanVec in support of OpenStreetMaps. The second-place contributor, *jfd553*, claims on his user page that he developed the conversion process to use CanVec data in OpenStreetMaps (OSM.) He says, "mapping the globe is like eating a whale, you do it one bite at a time!" Although CanVec is a high-quality data source, the OSM wiki warns inexperienced users to steer clear of using it. Importing duplicate data is a common mistake.

Quebec is composed of administrative regions, each containing 'urban agglomerations' consisting of independent municipalities or boroughs (Fr. *arrondissements*.) Phew. There is even a *Montreal Agglomeration Council* to govern the ongoing restructuring of political divisions. This means a "Montrealer" may identify his/her city as Montreal, Laval, or the island-suburb of Dorval and all of them are correct. Some users do make use of the "suburb" field which makes the data more accurate. Using a field labeled "administrative region" (which includes Montreal and Laval, the two most common entries under city) would introduce more clarity to the data. There also a possibility for confusion among French-speaking OSM users given that the field names are in their second language.

Geospatial indexing is an interesting feature of MongoDB that I would like to explore. Mapzen provides GeoJSON files for most cities, a file type which chops the data into lines, points, and polygons instead of the usual 'node, way, relation' format, permitting calculations on a sphere. It would be useful to cross-reference files with both a name and coordinate against files with just coordinates. The missing name could be updated with the name of the point at the matching coordinates. FedEx uses an algorithm with geospatial indexing to map the shortest route for a driver given his assigned stops. This idea could be applied to an app which shows the shortest route to a public toilet plotted in OSM.