Project 3: OpenStreetMap Data Wrangling with MongoDB

Jason Cabral

Map Area: Greater New Haven, CT
http://www.openstreetmap.org/#map=14/41.3084/-72.9170

1. Problems in the Data

In order to find opportunities for cleaning, I went through each set of attributes of the "tag" children of each element. Grouping them and counting each instance of the value of each tag, I was able to determine if there were formatting inconsistencies or invalid values for the particular tag. Many of the problematic tags were either fixed or eliminated programmatically and I used the OpenStreetMap documentation as my guide for how to proceed with each problem. A few examples are listed below.

a. Address Uniformity

Upon inspection of the address data, I found inconsistencies with house numbers, postcodes, street names, and state names. Although there is no consensus among the community on how to represent addresses with multiple values for house numbers, I decided to convert each list to a range to ensure uniformity across the values.

```
<tag k="addr:housenumber" v="91,93,95"/>
<tag k="addr:housenumber" v="39,37"/>
```

Tags such as those above were given values such as the following:

```
<tag k="addr:housenumber" v="91-95"/>
<tag k="addr:housenumber" v="39-37"/>
```

Next, any postcode with more than the standard 5 digits was trimmed to the proper 5 digit code. Some examples of problem codes are listed below.

```
<tag k="addr:postcode" v="06510-2404"/>
<tag k="addr:postcode" v="CT 06511"/>
```

Of the 318 'addr:state' tags, 26 had the full state name. According to the OpenStreetMap documentation, abbreviations are recommended for this tag, so the tags with the full name were converted to the abbreviation.

```
tag.set('v','CT')
```

Finally, I found problems with street names ranging from missing information to misspelling and improper abbreviation. These issues were corrected by replacing problems with their corresponding values in the following dictionary:

```
mapping = { "College": "College Street",
"Colledge Street": "College Street", "church": "Church","Blvd":
"Boulevard","Ave": "Avenue", "Rd": "Road", "St": "Street",
"street": "Street"}
```

b. Phone/Fax Number Uniformity

The phone and fax numbers in this dataset were presented with more than five different formats, some separated by non-numerical characters, others not separated at all. Remaining consistent with US standard, all numbers were converted to the following format 123-456-7890. Here are a few examples of numbers that did not follow this format:

+1 203 5625507
2035622511
203.776.5306
(203) 865-5762

c. Various Tag Issues That Were Addressed

A few other issues only occurred once or twice, but were clearly correctable, based on the OpenStreetMap documentation. Many of these errors are the result of outdated format or usage and were corrected to meet current standards. Below, you will find these problem tags and their fix.

`<tag k="capacity" v="undefined"/>` - Removed Tag
`<tag k="census:population" v="124001;2006"/>` - Split the year to it's own tag 'source:population' with value '2006'
`<tag k="maxspeed" v="30 mph;40 mph"/>` - Removed Tag
`<tag k="website" v="fleurdelysfloral.com"/>` - Added 'http://www.' for Uniformity
`<tag k="place_name" v="New Haven"/>` - Set Tag to k="name"
`<tag k="wood" v="deciduous"/>` - Set Tag to k="leaf_cycle"
`<tag k="drinkable" v="no"/>` - Set Tag to k="drinking_water"
`<tag k="Direct mailing" v="Direct Mail Marketing"/>` - Removed Tag

d. Tag Issues That Were Not Addressed

A few more tags had major issues with uniformity, consistency or clarity. In particular, I found the following tags especially problematic:
k="exit_to"
k="opening_hours"
k="source"

Exit_to=* is no longer a preferred tag as destination=* has taken it's place. Destination=*, opening_hours=* and source=* all have some sort of syntax and/or formatting guideline, but because these values include machine and human entered data and have a high degree of variability, correcting them programmatically would be unnecessarily difficult. It would be more efficient to manually correct these issues on an individual basis.

2. Data Overview

File sizes

New Haven Data.osm ..... 54.34 MB
Clean New Haven Data.json .... 61.44 MB

```
# Number of documents
db.new_haven.find().count()
```

```
287904
```

```
# Number of nodes
db.new_haven.find({"type":"node"}).count()
```

```
254782
```

```
# Number of ways
db.new_haven.find({"type":"way"}).count()
```

```
33122
```

```
# Number of unique users
db.new_haven.distinct("created.user").__len__()
```

```
176
```

```
# Number of entries per year
db.new_haven.aggregate([{"$project":{"_id":"$_id", "year":{"$substr":
["$created.timestamp",0,4]}}},{"$group" : {"_id": "$year", "count":
{"$sum":1}}}, {"$sort" : {"count" : -1}}]):
```

```
# “_id” represents the year of entry
{u'_id': u'2015', u'number_of_entries': 192706}
{u'_id': u'2014', u'number_of_entries': 49451}
{u'_id': u'2013', u'number_of_entries': 30508}
{u'_id': u'2012', u'number_of_entries': 6344}
{u'_id': u'2009', u'number_of_entries': 4521}
{u'_id': u'2010', u'number_of_entries': 2544}
{u'_id': u'2011', u'number_of_entries': 1551}
{u'_id': u'2008', u'number_of_entries': 278}
{u'_id': u'2007', u'number_of_entries': 1}
```

One of the keys to an accurate mapping program is making sure it's data is up to date. 84% of this data has been entered or updated in the last two years and while this does not completely eliminate our concern of outdated information, the high percentage of recently entered data should give us some level of confidence about its accuracy.

```
# Top Contributor
db.new_haven.aggregate([{"$project":{"_id":"$_id",
"contributor":"$created.user"}}, {"$group" : {"_id": "$contributor",
"count":{"$sum":1}}},{"$sort" : {"count" : -1}},{"$limit" : 1}])
```

```
{'count': 156810, '_id': 'redsteakraw'}
```

```
# Number of users appearing only once (having 1 post)
db.new_haven.aggregate([{"$project":{"_id":"$_id",
"contributor":"$created.user"}}, {"$group" : {"_id": "$contributor",
"count":{"$sum":1}}},{"$match":{"count":{"$lt":2}}}]):
```

```
43
```

3. Other Ideas or Observations About the Dataset

```
# New Haven Pizza
```

New Haven is world famous for it's pizza, so its no surprise that Pizza and Italian restaurants account for 35% of the cuisine types recorded in the dataset.

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

```
{u'count': 15, u'_id': u'pizza'}
{u'count': 9, u'_id': u'italian'}
```

```
# Yale University
```

New Haven is also well known for being the home of Yale University, however their reach goes beyond the school itself. The code below explores how many documents contain “Yale” in either the 'name' or 'operator' tag.

```
name = db.new_haven.aggregate([{"$match" : {"name.name":{"$exists":1}}},
{"$project" : {"_id" : "$_id", "name" : "$name.name"}}])
operator = db.new_haven.aggregate([{"$match" : {"properties.operator":
{"$exists":1}}}, {"$project" : {"_id" : "$_id", "operator" :
"$properties.operator"}}])
```

```
601
```

```
# Roads That Span The Most Zip Codes
```

```
db.new_haven.aggregate([{"$match":{"tiger":{"$exists":1}}}, {"$match":
{"highway":{"$exists":1}}},{"$match":{"name.name":{"$exists":1}}},
{"$project": {"_id":"$name.name", "zip_1":"$tiger.zip_left",
"zip_2":"$tiger.zip_left_1","zip_3":"$tiger.zip_left_2","zip_4":"$tiger.
zip_right","zip_5":"$tiger.zip_right_1","zip_6":"$tiger.zip_right_2"}}])
```

```
High Street : 3
set([u'06511', u'06512', u'06516'])
Bradley Street : 3
set([u'06511', u'06510', u'06512'])
```

Concluding Thoughts

Many of the problems I encountered with regard to the quality of the data were related to inconsistencies in formatting, missing information or the use of improper and outdated tags. For a mapping program such as this to be considered successful, it is vital that the information available is complete, accurate and consistent. Furthermore, as the number of applications that use this data for navigation purposes grows, so too does the importance for clean data.

Two ways to ensure more complete data are to require tags for certain entries or automatically tag entries based on their existence within a 'relation' (some sort of boundary). While examining the New Haven dataset, I found that only 0.56% of buildings had a full address that included a zip code. One example of how to fix this would be to require a street name and city for each building be entered by the editor. Another example of a way to ensure more complete data is to automatically tag any building within the New Haven relation with a city tag, valued 'New Haven'. Both should be reasonably simple additions to the OpenStreetMap codebase and could either be invisible to the end user or implemented by way of auto-filling values into a map editor. The benefit to this is that the data will be more complete and entries would be consistently up to date for a particular tag. However, the biggest problem I can see with this implementation is with contested or changing borders. While the borders in my dataset were fairly straightforward, there are areas of the world that can not be so easily defined by a simple 'relation' border. For this improvement to be successful globally, it would need to be flexible enough as to not make changes automatically in contested regions.

Next, I found many issues with value formatting, especially with road names and phone numbers. Examples of both and how they were cleaned are shown in the first section of this document. OpenStreetMap data is being used in a large number of third party mapping programs, including some for navigation, so the consistency of this data is important to those who are seeking assistance from these programs. With regard to map editors, an easy solution would simply be to have drop menus with valid street types and phone number boxes with specific format requirements. However, this becomes more troublesome for imported data as the code can not possibly account for all potential human entered formats. Additionally, one concern I found during cleaning was that all streets in a particular region may not follow convention. For example, while I was expecting a value of Street, Ave., Rd. or Court at the end of each street name, 'Broadway' failed the test while at the same time being a completely valid value.

One improvement I devised seems to have very little downside and that is the standardization of tags in the database. I found a number of tags in the New Haven data that were either not valid tags according to the OpenStreetMap documentation or tags that had been deprecated by the community and changed to or merged with a different tag. Changing problematic tags for which there is an agreed resolution should be a simple programmatically. In fact, the OpenStreetMap wiki tracks the number of uses for each tag in the database, so updating them programmatically, should be a simple process. One example as shown earlier in the document is simple converting 'k=wood' to 'k=leaf_cycle'. The values of this tag would not need to be altered in any way. I contend that any tag that does not exist within the OpenStreetMap framework be deleted from the database entirely. One concern is that valuable information could be lost, but because the tags are unknown, the only way to correct them would be for a human to go through them individually or to simply delete them altogether in order to avoid confusion for end users.

Finally, in an attempt to address the previous concern, OpenStreetMap could introduce more robust editor collaboration and recognition. In an attempt to improve it's status as an authority on local information, Google has recently introduced an incentive program, called Local Guides, to encourage more people to generate data within it's ecosystem. While this type of program is likely outside of the scope or budget of OpenStreetMap, it shows how valuable user-generated data can be to the success of a mapping program. And while any human generated data will be prone to error, it is sometimes only a human eye that can correct

those errors. Waze, a mapping and navigation platform which was recently purchased by Google, has been built from the ground up on user-generated data by using simple gamification techniques that do not cost any money but encourage collaboration and interaction.

To this end, OpenStreetMap could use a flag system to mark particular entries or tags as questionable. Editors could register or be recognized as local authorities for a particular region and be alerted to these flags. Foursquare found success by recognizing frequent visitors as “Mayors” of a particular area or establishment and SeeClickFix allows owners of businesses to respond to user generated concerns that are flagged within a defined area around their business. I believe that similar features could be implemented at little cost to OpenStreetMap. In fact, given that the data itself is available to the public, a third party application developer may be able to incorporate some of these ideas and then have the corrected information uploaded back to the database.

Given the vast amount of data contained within OpenStreetMap's databases, make large improvements could seem daunting or costly. However, I believe that implementing a few small changes, as explained above, would go a long way to improving the data that end users find most important.

4. References

Documentation:
https://docs.python.org/2/library/xml.etree.elementtree.html#xml.etree.ElementTree.iterparse
https://docs.python.org/2/library/re.html
https://docs.python.org/2/library/string.html
http://effbot.org/zone/element-index.htm
https://docs.mongodb.org/v3.0/reference/

Discussion:
http://stackoverflow.com/questions/18796280/set-attribute-to-element-in-python
http://stackoverflow.com/questions/4788633/insert-a-node-for-an-element-in-xml-with-python-elementtree
https://discussions.udacity.com/t/importing-the-clean-json-file-into-a-mongodb-database/16712
https://discussions.udacity.com/t/error-in-distinct-query-in-mongodb/32590/7
http://stackoverflow.com/questions/16772071/sort-dict-by-value-python