Udacity Data Analyst Nanodegree Project 2

Brian Hurn
May 30, 2015 (revised)

Overview of the data

For this project, I selected a portion of Cobb County, Georgia, in the United States as my area of interest. I chose this particular area because it includes my residence, and I felt that having some local knowledge might prove useful (which it did).

Using the "What's Here?" right-click function in Google Maps, I selected two points (33.81, -84.50 and 34.08, -84.74) that defined a rectangular bounding box in western Cobb County consisting of approximately 665 square kilometers. Utilizing the Overpass API, I retrieved an "interpreter" file that was 72.7 MB in size.

The general area in question and an outline of Cobb County can be seen here:

> http://www.openstreetmap.org/relation/1020125#map=11/33.9448/-84.5734

Data wrangling process

My first step was to explore the OSM file using the `xml.etree.ElementTree` library techniques introduced in the course.

A count of the various tags in the file yielded this result:

```
{'member': 11259,
 'meta': 1,
 'nd': 417383,
 'node': 326246,
 'note': 1,
 'osm': 1,
 'relation': 288,
 'tag': 239357,
 'way': 28398}
```

In addition, I discovered that the number of unique users that contributed to the node, way, and relation entries was 234. I also listed and checked the number of unique *node* attributes (8), unique *way* attributes (6), and unique values for *k* in *tag* tags (386).

Next, I checked the zip codes found in the *addr:postcode* tags. In all, 29 zip codes were present. I manually checked the location of each of these zip codes (again leveraging my familiarity with Google Maps), to confirm that they indeed lie within the bounding box for the data. I discovered that two values (30092 and 30144) are assigned to municipalities that lie outside of Cobb County and the area of interest. This points to a data entry problem introduced by one or more contributors. Later I describe the MongoDB queries that I used to explore this issue further.

The relevant portion of the code that I used to develop the list and count of the zip codes is shown below. The sorted() function is especially helpful in generating legible output.

```
def process_map(filename):
      zips = set()
      for _, element in ET.iterparse(filename):
            if element.tag == "tag" and "k" in element.attrib.keys():
                  if element.attrib["k"] == "addr:postcode":
                        zips.add(element.attrib["v"])

      return zips

zips = process_map(OSMFILE)
print "Unique zipcodes: {}".format(len(zips))
for zip in sorted(zips):
      print zip
```

One of the points of emphasis in the course, and rightly so, is the quality of the street names in the OSM data. I used the code shown in *street_audit.py* to print a list of the street types that could not be found in the initial *expected* list used in the *prepare_xml.py* program that generated the JSON file that would serve as the input to MongoDB. Before the first pass I had added to the list certain street types that I know can be found in this area, including my own, “Way.” I pursued an iterative process to ensure that all acceptable types were included in the *expected* list (and therefore not modified by *prepare_xml.py)*.

This audit revealed some expected issues (such as “St” and “St.”, which should be “Street”). These issues were easy to handle programmatically. The audit also showed an entry for “4015”, which was part of the suite number for a dentist’s office that had been included in the street address. These types of unique situations are best handled manually.

The most significant issue that I found was the use of directional suffixes such as “Southwest”, “Southeast”, etc. in street names (e.g. “Green Street Southeast”). This is common practice in this area, and my understanding is that these terms indicate the quadrant or half of the zip code in which the address lies. My own official postal address in fact ends in “SW”.

As a first step, I determined, based on the official addresses that the Postal Service provides for this area, that the directional suffixes should all be reduced to their one- or two-letter abbreviations ("S", "SW", "SE", etc.), and I updated the *expected* and *mapping* variables accordingly. The issue of potential improper street types (e.g. "Rd" vs "Road") in the word preceding the directional suffix remained, however. This required an additional step in my code to process the next to last word in the same manner as the last word for streets without the suffix. This check was only performed if the first pass produced a final word equal to one of the one- or two-letter directional abbreviations. Several MongoDB queries confirmed that these steps was performed properly.

Overview of the data using MongoDB

After inserting the JSON file from the previous step into MongoDB using *mongo_insert.py*, I explored the data using the individual queries shown in the *mongo_queries.py* file. The MongoDB aggregate pipelines and findings are as follows:

User with most entries

```
pipeline = [{"$group": {"_id": "$created.user", "count": {"$sum": 1}}},
            {"$sort": {"count": -1}},
            { "$limit": 1}]
```

yielded the following:

```
{u'_id': u'Liber', u'count': 93820}
```

This is a surprising result given the huge number of entries. I imagine that some automated tools were used to generate this much content.

Number of unique users (for node and way entries)

```
pipeline = [{"$group": {"_id": "$created.user", "count": {"$sum": 1}}},
            {"$group": {"_id": "_id", "count": {"$sum": 1}}}]
```

yielded the following:

```
{u'_id': u'_id', u'count': 225}
```

This number is slightly smaller than the total that I found earlier in the XML (234), but that value included relations as well.

Amenity counts

```
pipeline = [{"$group": {"_id": "$amenity", "count": {"$sum": 1}}},
            {"$sort": {"count": -1}}]
```

produced this result (first ten shown for brevity):

```
{u'_id': None, u'count': 353146}
{u'_id': u'parking', u'count': 348}
{u'_id': u'place_of_worship', u'count': 200}
{u'_id': u'grave_yard', u'count': 134}
{u'_id': u'school', u'count': 121}
{u'_id': u'restaurant', u'count': 109}
{u'_id': u'fuel', u'count': 101}
{u'_id': u'fast_food', u'count': 81}
{u'_id': u'atm', u'count': 72}
{u'_id': u'bank', u'count': 55}
```

Considering the volume of entries indicating “None,” one can see that it would be useful to encourage as much use of this field by contributors as possible. It’s interesting to note the quantity of parking locations documented. Given the suburban nature of this area, finding a parking place is not typically an issue, but it would be useful for users to understand what their options are for a range of destinations. Unfortunately, the Open Street Map key does not appear to indicate parking areas at this time.

Problem zip code counts

```
pipeline = [{"$match": {"address.postcode": {"$in": ["30092", "30114"]}}},
            {"$group": {"_id": "$address.postcode", "count": {"$sum": 1}}}]
```

yielded:

```
{u'_id': u'30092', u'count': 20}
{u'_id': u'30114', u'count': 1}
```

As a final step, I searched to see the extent of of issues with the problem zip codes (those that fall outside the area of interest) found earlier. The 20 entries for 30092 were all created by a single user and fall within the 30082 zip code, so this was simply a typographical error that could easily be corrected.

Other ideas

This dataset provides a number of opportunities for additional exploration. As an example, one could explore the user data and find high-volume users that have delivered high-quality input with limited errors over an extended period of time. These users could be contacted by the OpenStreetMap team and offered roles as reviewers, evangelists, or trainers. In order to complete this type of analysis, it would be necessary to develop quality standards (and perhaps a scoring system) as well as an approach that tracks engagement and quality over time through periodic audits. This could be time-consuming and unwieldy given the extent of the OpenStreetMap footprint.

As another example, this data could be explored for business-to-business marketing purposes. Using the geospatial capabilities of MongoDB, one could find all the instances of a given type of business within a certain radius of a chosen location. These businesses could then be mined as prospects for B2B sales opportunities. Two potential problems come to mind. The first is data quality in terms of accuracy and completeness; many businesses have not yet been identified and entered, and the open source nature of OpenStreetMap and its relatively small user base don't point to a ready solution. The second is the size of the data the necessary to perform this analysis at scale, which could lead to computer memory or storage challenges for a typical user.

References

Stackoverflow.com: IndexError: list assignment index out of range; json.loads shows ValueError; Parsing values from a JSON file in Python; Pymongo - cursor iteration; How to update values using pymongo?

MongoDB manual: Aggregation; Count; Group; Tutorial; $in