Wrangle OpenStreetMap Project

January 24, 2016

1 Udacity Data Wrangling with MongoDB

1.0.1 Wrangle OpenStreetMaps Project - Greg Hein

The map data is taken from MapZen:
https://mapzen.com/data/metro-extracts
The data chosen to investigate is the OSM XML data for Milwaukee, WI, United States:
https://www.openstreetmap.org/relation/251075
That data is avaliable here:
https://s3.amazonaws.com/metro-extracts.mapzen.com/milwaukee_wisconsin.osm.bz2
The data was analyzed by importing the OSM XML file found above, cleaning and converting to JSON. That JSON file was then loaded into a MongoDB Database, and was analyzed using MongoDB queries with the help of the pymongo package for the Python programming language.

1.1 1. Problems in the Data Set

1.1.1 A. Multiple Street Type Abbreviations

A condensed version of the data was created by filtering out all data fields that did not contain address information. After visually inspecting that file, it was determined that there were multiple instances where the street type (eg. Avenue, Street, Boulevard, Road) was inputted with multiple types of abbreviations in the data set. The data was cleaned to remove all abbreviations and replace with the entire word.

1.1.2 B. Multiple Compass Direction Abbreviations

Also, when inspecting the condensed file, it became evident there was a similar problem to the street type problem listed above. Specifically many street names began with a compass direction (North, South, East, and West) but in many cases those were abbreviated, and not consistently. The data was cleaned to remove all of the abbreviated compass directions and replace them with the entire word.

1.1.3 C. Minimal Detail in Data

A visual inspection of portions of the large datafile seemed to suggest that most nodes in the database were strictly there to store latitude and longitude, most likely to help define the ways in the database. Those nodes did not look to be storing more detailed information, such as address fields, building fields, amenity fields, etc.

Some queries were run to see if this was the case:

```
In [6]: from pymongo import MongoClient
        import pprint
        client = MongoClient('localhost:27017')
        db = client.streetmap

        print "Number of nodes:", db.mke.find({'type':'node'}).count()
        print "Number of data entries with position info:", \
```

```
        db.mke.find({'pos': {'$exists': True}}).count()
print "Number of data entries with address info:", \
        db.mke.find({'address': {'$exists': True}}).count()
print "Number of data entries with street info:", \
        db.mke.find({'address.street': {'$exists': True}}).count()
print "Number of data entries with amenity info:", \
        db.mke.find({'amenity': {'$exists': True}}).count()
print "Number of data entries with building info:", \
        db.mke.find({'building': {'$exists': True}}).count()
```

```
Number of nodes: 697799
Number of data entries with position info: 697806
Number of data entries with address info: 2804
Number of data entries with street info: 2317
Number of data entries with amenity info: 5065
Number of data entries with building info: 9445
```

Having less than 10,000 nodes in each of these more detailed fields in a database with almost 700,000 nodes indicated very sparse data with respect to this more detailed information.

1.2 2. Overview of the Data

This section will give overview data and information about the database.
The MongoDB queries and pymongo code used to gather this information is also given.

The size of the the original OSM XML file:

```
mke.osm - 153,215 KB
```

The size of the converted JSON file:

```
mke.json - 173,358 KB
```

General Data statistics:

```
In [7]: print "Number of data entries in database:", db.mke.count()
        print "Number of unique users:", len(db.mke.distinct("created.user"))
        print "Number of nodes:", db.mke.find({'type':'node'}).count()
        print "Number of ways:", db.mke.find({'type':'way'}).count()
```

```
Number of data entries in database: 776300
Number of unique users: 626
Number of nodes: 697799
Number of ways: 78486
```

The different tags in the database were found with the program CountTags.py:

```
In [12]: import xml.etree.cElementTree as ET

         def count_tags(filename):
                 tags = {}

                 for event, elem in ET.iterparse(filename, events = ('start',)):
                     if elem.tag not in tags:
                         tags[elem.tag] = 1
                     else:
                         tags[elem.tag] = tags[elem.tag] + 1
```

```
            return tags

    tags = count_tags('mke.osm')
    pprint.pprint(tags)
```

```
{'bounds': 1,
 'member': 6579,
 'nd': 859194,
 'node': 697806,
 'osm': 1,
 'relation': 540,
 'tag': 443856,
 'way': 78494}
```

Some aggregation queries were run to determine information about the makeup of the users contributing to the data:

```
In [9]: ##    Aggregation queries:

    def aggregate(db, pipeline):
        return [doc for doc in db.mke.aggregate(pipeline)]

    print '\nNumber of users with only one contribution:'
    pipeline1 = [{"$group": {"_id": "$created.user", "count": {"$sum": 1}}},
                 {"$group": {"_id": "$count", "num_users": {"$sum": 1}}},
                 {"$sort": {"_id": 1}},
                 {"$limit": 1}]
    result1 = aggregate(db, pipeline1)
    pprint.pprint(result1)

    print '\nTop 5 contiributing users:'
    pipeline2 = [{"$group": {"_id": "$created.user", "count": {"$sum": 1}}},
                  {"$sort": {"count": -1}},
                  {"$limit": 5}]
    result2 = aggregate(db, pipeline2)
    pprint.pprint(result2)
```

```
Number of users with only one contribution:
[{u'_id': 1, u'num_users': 122}]

Top 5 contiributing users:
[{u'_id': u'woodpeck_fixbot', u'count': 181244},
 {u'_id': u'ItalianMustache', u'count': 69763},
 {u'_id': u'reschultzed', u'count': 45951},
 {u'_id': u'Gary Cox', u'count': 32553},
 {u'_id': u'bbauter', u'count': 31861}]
```

1.3 3. Other Additional Ideas

1.3.1 A. Ways to improve the data

One area for improvement to the database would be the addition of more detail to the data. Modern maps not only give street and highway directions, but it is commonly expected that certain amenities and other local information are given too. Some examples a traveler or visitor to a new area would be interested in include: restaurants, museums, zoos, golf courses, night clubs, places of worship, etc. The Milwaukee

metropolitan area consists of greater than 2 million people (https://en.wikipedia.org/wiki/Milwaukee), so it would seem obvious that there are many more than 5065 amenities in the area.

One way to remedy this problem would be encourage those who own and/or run such establishments to enter that information into OpenStreetMaps themselves. It would seem obvious that it would be in their self interest as a possible way to increase awareness and access to their shops/churches/etc. With OpenStreetMaps being an open project contribution is essentially free to someone only entering one or two data entries. The problem to overcome is increasing awareness of the project. Typically the best way to increase awareness is through advertising, but the ability to spend money to promote an open project is likely to be minimal, so that is unlikely an option.

Another way to remedy that problem would be to gather data from some outside source. The obstacle there is finding a source that is not proprietary.

1.3.2 B. Additional data exploration using MongoDB aggregation queries:

```
In [13]: print '\nTop 5 types of restaurant cuisine categories:'
         pipeline3 = [{"$match": {"amenity": {"$exists": 1}, "cuisine": \
                                  {"$exists": 1}, "amenity": "restaurant"}},
                      {"$group": {"_id": "$cuisine",  "count": {"$sum": 1}}},
                      {"$sort": {"count": -1}},
                      {"$limit": 5}]
         result3 = aggregate(db, pipeline3)
         pprint.pprint(result3)

         print '\nTop 5 types of buildings:'
         pipeline4 = [{'$match': {'building': {'$exists': 1}}},
                      {'$group': {'_id': '$building', 'count': {'$sum': 1}}},
                      {'$sort': {'count': -1}},
                      {'$limit': 5}]
         result4 = aggregate(db, pipeline4)
         pprint.pprint(result4)

         print '\nTop 5 types of amenities:'
         pipeline5 = [{'$match': {'amenity': {'$exists': 1}}},
                      {'$group': {'_id': '$amenity', 'count': {'$sum': 1}}},
                      {'$sort': {'count': -1}},
                      {'$limit': 5}]
         result5 = aggregate(db, pipeline5)
         pprint.pprint(result5)
```

```
Top 5 types of restaurant cuisine categories:
[{u'_id': u'american', u'count': 29},
 {u'_id': u'pizza', u'count': 28},
 {u'_id': u'italian', u'count': 23},
 {u'_id': u'chinese', u'count': 15},
 {u'_id': u'mexican', u'count': 14}]

Top 5 types of buildings:
[{u'_id': u'yes', u'count': 5963},
 {u'_id': u'house', u'count': 755},
 {u'_id': u'apartments', u'count': 592},
 {u'_id': u'commercial', u'count': 458},
 {u'_id': u'garage', u'count': 356}]

Top 5 types of amenities:
[{u'_id': u'parking', u'count': 1800},
```

```
{u'_id': u'school', u'count': 905},
{u'_id': u'restaurant', u'count': 385},
{u'_id': u'fast_food', u'count': 202},
{u'_id': u'fuel', u'count': 164}]
```

1.3.3 C. Conclusion

As is likely to be expected in an open database project, this data is raw and incomplete. Although the Milwaukee metropolitan area is relatively large (5th largest in the Midwest United States: https://en.wikipedia.org/wiki/Milwaukee) much of the detailed data common to many modern maps is missing. It will be interesting to see how the project progresses in the future.