Project 3

Data Wrangling with Mongo DB

08/13/15

Name : Rajiv Kumar

Contents

1. Problems Encountered in the Map:

I live in Aurora-Colorado and I wanted to explore my city data.So i choose to use Aurora-Co city map data from openstreet map data sets. I was able to download the full city data, which was ~72 MB (size of osm file). After reviewing the data, I found following problems with it:

- Cities other than “Aurora” in the .osm file.
- Postal codes , which do not belong to city Aurora,CO
- Tiger Mapped data.
- gnis mapped data:
- Non uniformity in the street names

- Non uniformity in the state name
- Non uniformity in county name k value

I will describe how I cleaned the data and removed the non-uniformity and inconsistency from the data in the following sections.

Data Cleaning:

Cities other than "Aurora" in the .osm file:

The downloaded .osm file had data for neighbor cities, which are not part of Aurora. For example , cities such as centennial , parker etc.Any analysis done on this data without knowing that the data contained other cities could have given incorrect statistics. So, I removed all the nodes that contained data for other cities. To do this, I parsed through the tag and find k value city and then I filtered all the records for which city was not equal to Aurora. You can find more details in the code file.

Postal codes , which do not belong to city Aurora,CO:

I noticed that for several nodes, the post code was not Aurora's post code. These addresses were addresses from adjacent cities. I pulled out the list of Aurora's post codes from following website http://www.geonames.org/postalcode-search.html?q=aurora&country=US&adminCode1=CO .I put this list of postal codes in a list inside my python program for reference. I basically used cross checking to verify post codes in the data and filtered out those records , which have post codes other than the ones present in the list

Tiger Mapped data:

While reviewing the data , I came across following tag values.

```
<tag k="tiger:cfcc" v="A41"/>
<tag k="tiger:county" v="Arapahoe, CO"/>
<tag k="tiger:name_base" v="Dry Creek"/>
<tag k="tiger:name_direction_prefix" v="E"/>
<tag k="tiger:name_type" v="Rd"/>
<tag k="tiger:zip_left" v="80016"/>
<tag k="tiger:zip_right" v="80016"/>
```

These tags , if not cleaned properly were going to have somewhat different format for key values in my final JSON data format.I decided to research these tags in the open street maps documentation. From my research , I found that

The Topologically Integrated Geographic Encoding and Referencing system (TIGER) data, produced by the US Census Bureau, is a public domain data source which has many geographic features. The TIGER/Line files are extracts of selected geographic information, including roads, boundaries, and hydrography features. All of the roads were imported into OSM in 2007 and 2008, populating the nearly empty map of theUnited States.

In my further research, I found that these tags represent mapping with the osm fields as described in the following document for openstreet map documentation.
http://wiki.openstreetmap.org/wiki/TIGER_to_OSM_Attribute_Map

In order to organize this data well in my json format, I decided to create a dictionary Tiger inside my node dictionary and put all the key values of tiger data set inside the dictionary after removing the tiger: from the key value.

Later,o I noticed that tlid attribute had list of values separated by ‘:’ inside the tiger dictionary . I decided to bring these values in an array(‘tlid’) separated by ‘,’ inside the Tiger dictionary. Tiger dictionary looked like following at the end:


```
"tiger": {
   "separated": "no",
   "name_base": "Buckboard",
   "zip_left": "80138",
   "tlid": [
     "130803656",
     "130803661"
   ],
   "cfcc": "A41",
   "reviewed": "yes",
   "county": "Douglas, CO",
   "source": "tiger_import_dch_v0.6_20070809",
   "name_direction_prefix": "E",
   "name_type": "Rd",
   "zip_right": "80138"
```


gnis mapped data:

During my further analysis , I found that there exists another mapping called gnis in some of the tags. I went back to the open street maps documentation to know more about this mapping and I found that

The USGS Geographic Names Information System (GNIS) is a database that contains millions of names for geographic features in the United States and Antarctica. The system is run by the

Board of Geographic Names, a United States Geological Survey group. It is the authoritative set of geographic names for the US. It contains features that are on no other map or spatial database.

GNIS US data was bulk-imported in 2009 into OSM, and vast swathes of incorrect data still needs to be tracked down and corrected. The fundamental problem with GNIS is that it is a database of "names" and not "features" - if you want to answer the question "where is/was Foobar School" then GNIS will have a coordinate for that name and know that it is/was a school. Unfortunately GNIS records were imported into OSM without regard as to whether or not those features still exist, so there are tens of thousands of churches, schools, etc., that have long since disappeared - pre-dating the Interstate system in many (obvious) cases.

I decided to give this tag, the same treatment as I gave to tiger: data so I created a separate dictionary for gnis mapped data under my node dictionary.

Non uniformity in the street names :

Some of the non-uniformities were already addressed in chapter 6 exercise. However, I analyzed the street names in the AURORA city data to ensure that there are no more such uniformities. During my research , I came across following additional non uniformities in the data.

- Street names with “E.” or “East”
- Street names starting with “S” , “S.” or “South”
- Street names ending with Rd

I modified my python program and changed the street names to bring uniformity by doing following.

- Changed all the street name starting with E. to East
- Street names starting with “S” , “S.” to “South”
- Changed Street names ending with Rd to Road

I followed the same approach to modify these street names as we did in chapter 6 exercise by adding these fields in the mapping dictionary.

Non uniformity in the State name:

In my further analysis, I found that state name in the address was not uniform. At some places, the state abbreviation was used and at other places , full state name was used. I fixed this non-uniformity by changing state name to full name “Colorado”

Non uniformity in county name k value :

In my further analysis, I discovered that k value for County name in the data was not uniform. At some places ***county_name*** was used as k value and at other places ***county*** was used to describe county names. I modified all the records so that all records contain ***county*** to describe county names.

After all my filtration, I had to discard data of size 213 KB. These data was mainly from other cities. It did not make this data in the data set as it would have created troubles during any analysis.

Data Overview:

This section contains some statistics about the data and MONGODB queries;

File Sizes:

Map.osm----------72MB

Map.json----------79MB

Number of records:

- # of Total Records: 358116
  - `db.Aurora.find().count()`
- # of Records with type=Node: 327127
  - `db.Aurora.find({'type':'node'}).count()`

- # of Records with type=way : 30984
  - `db.Aurora.find({'type':'way'}).count()`
- # of unique users : 239
  - `db.Aurora.distinct("created.user").length`

Additional Data Insights:

- Top 3 cuisines by type
  - `{ "_id" : "american", "count" : NumberInt(22)}`
  - `{ "_id" : "mexican", "count" : NumberInt(14)}`
  - `{ "_id" : "asian", "count" : NumberInt(9)}`
  - `db.Aurora.aggregate([{"$match":{'cuisine':{"$exists":1},"amenity":"restaurant"}},{"$group":{"_id":"$cuisine","count":{"$sum":1}}},{"$sort":{"count":-1}},{"$limit":3}])`
- Number of nodes that have tiger data : 6583
  - `{"_id" : "tiger", "count" : NumberInt(6583)}`
  - `db.Aurora.aggregate([{"$match":{'tiger':{"$exists":1}}},{"$group":{"_id":"tiger","count":{"$sum":1}}}])`
- Number of nodes that have gnis data :
  - `{"_id" : "tiger", "count" : NumberInt(151)}`
  - `db.Aurora.aggregate([{"$match":{'gnis':{"$exists":1}}},{"$group":{"_id":"tiger","count":{"$sum":1}}}])`
- #1 contributor user :
  - `{"_id" : "Your Village Maps","count" : NumberInt(135372)}`
  - `db.Aurora.aggregate([{$group:{_id:"$created.user","count":{$sum:1}}},{$sort:{count:-1}},{$limit:1}])`
- Highest number of restaurant in the postcode:

`{ "_id" : "80014", "count" : NumberInt(50)}`

- `db.Aurora.aggregate([{$match:{amenity:"restaurant",address:{$exists:1}}},{$group:{_id:"$address.postcode",count:{$sum:1}}},{$sort:{"count":-1}},{$limit:1}])`

Note: we are only considering those nodes that have address field so this data might not be accurate

- Top 2 years with maximum contribution based on timestamp

  - {"_id" : {"year" : "2013" }, "count" : NumberInt(89821)}
  - {"_id" : { "year" : "2015"}, "count" : NumberInt(83551)}

  - db.Aurora.aggregate([{$group:{"_id":{year:{$substr:["$created.timestamp",0,4]}},"count":{$sum:1}}},{$sort:{"count":-1}},{$limit:2}])

- List of amenities by their respective count in decreasing order:

```
{     "_id" : "parking",     "count" : NumberInt(1857)}
{     "_id" : "restaurant",     "count" : NumberInt(164)}
{     "_id" : "fast_food",     "count" : NumberInt(110)}
{     "_id" : "school",     "count" : NumberInt(104)}
{     "_id" : "shelter",     "count" : NumberInt(94)}
{     "_id" : "fountain",     "count" : NumberInt(83)}
{     "_id" : "fuel",     "count" : NumberInt(57)}
{     "_id" : "bank",     "count" : NumberInt(56)}
{     "_id" : "swimming_pool",     "count" : NumberInt(56)}
{     "_id" : "place_of_worship",     "count" : NumberInt(50)}
{     "_id" : "toilets",     "count" : NumberInt(44)}
{     "_id" : "car_wash",     "count" : NumberInt(38)}
{     "_id" : "bench",     "count" : NumberInt(32)}
{     "_id" : "cafe",     "count" : NumberInt(24)}
{     "_id" : "pharmacy",     "count" : NumberInt(18)}
{     "_id" : "fire_station",     "count" : NumberInt(17)}
{     "_id" : "waste_basket",     "count" : NumberInt(14)}
{     "_id" : "bicycle_parking",     "count" : NumberInt(14)}
{     "_id" : "atm",     "count" : NumberInt(13)}
{     "_id" : "public_building",     "count" : NumberInt(13)}
{     "_id" : "bar",     "count" : NumberInt(12)}
{     "_id" : "dentist",     "count" : NumberInt(11)}
{     "_id" : "kindergarten",     "count" : NumberInt(11)}
{     "_id" : "hospital",     "count" : NumberInt(9)}
{     "_id" : "pub",     "count" : NumberInt(9)}
{     "_id" : "doctors",     "count" : NumberInt(8)}
{     "_id" : "bbq",     "count" : NumberInt(7)}
{     "_id" : "post_box",     "count" : NumberInt(7)}
{     "_id" : "childcare",     "count" : NumberInt(6)}
{     "_id" : "post_office",     "count" : NumberInt(6)}
{     "_id" : "drinking_water",     "count" : NumberInt(5)}
{     "_id" : "police",     "count" : NumberInt(5)}
{     "_id" : "veterinary",     "count" : NumberInt(4)}
{     "_id" : "recycling",     "count" : NumberInt(4)}
{     "_id" : "cinema",     "count" : NumberInt(4)}
{     "_id" : "telephone",     "count" : NumberInt(4)}
{     "_id" : "theatre",     "count" : NumberInt(3)}
{     "_id" : "library",     "count" : NumberInt(3)}
```

```
{     "_id" : "community_centre",     "count" : NumberInt(2)}
{     "_id" : "clinic",     "count" : NumberInt(2)}
{     "_id" : "arts_centre",     "count" : NumberInt(2)}
{     "_id" : "parking_entrance",     "count" : NumberInt(2)}
{     "_id" : "nightclub",     "count" : NumberInt(2)}
{     "_id" : "bus_station",     "count" : NumberInt(2)}
{     "_id" : "car_rental",     "count" : NumberInt(2)}
{     "_id" : "university",     "count" : NumberInt(1)}
{     "_id" : "charging_station",     "count" : NumberInt(1)}
{     "_id" : "dojo",     "count" : NumberInt(1)}
{     "_id" : "vending_machine",     "count" : NumberInt(1)}
{     "_id" : "grave_yard",     "count" : NumberInt(1)}
    db.Aurora.aggregate([{$match:{"amenity":{$exists:1}}},{$group:{_id:"$amenity",

    "count":{$sum:1}}},{$sort:{count:-1}}])
```

Additional Ideas:

The analysis of data shows that most the data has been added after the year 2008 with following stats

- Percentage of Data before year 2008 =0.02 %
- Percentage of Data before year 2008 >98 %

Also,

- 25% ,which is significant percentage of the data, was added in the year 2013 only.

After American cuisine, Aurora has highest number of Mexican cuisine in the city.

User" your village maps" has the maximum contributions, which is 37 %..

While analyzing the data, I surfaced that some data does not actually belong to Aurora city. There should be a way to discover and filter all such data. However, it becomes increasingly difficult for those nodes, where address field is not present. In the Aurora city data set , only 812 nodes contain the address field. So for rest of the fields we can choose to use POS (longitude and latitude) attributes to check if the given point is within the Aurora Area (polygon) or not. This can be achieved using using **contains_point** function from **geopy.geocoders** library. Below is an example: reference : http://stackoverflow.com/questions/16625507/python-checking-if-point-is-inside-a-polygon

```
import matplotlib.path as mplPath
poly = [190, 50, 500, 310]
bbPath = mplPath.Path(np.array([[poly[0], poly[1]],
                    [poly[1], poly[2]],
                    [poly[2], poly[3]],
                    [poly[3], poly[0]]]))

bbPath.contains_point((200, 100))
```

Additionally, in order to make the data more useful we can derive the address fields from the longitude and latitude field . We can use Nominatim from geopy.geocoders library to derive the address from the longitude and latitude. Below is an example:

```
>>> from geopy.geocoders import Nominatim
>>> geolocator = Nominatim()
>>> location = geolocator.reverse("52.509669, 13.376294")
>>> print(location.address)

Alternatively, we can also use googlev3 from geopy.geocoders library to do the same.
Below is an example:

>>>from geopy.geocoders import GoogleV3
>>> geolocator = GoogleV3()
>>> address, (latitude, longitude) = geolocator.reverse("40.752067, -73.977578")
>>> print address, latitude, longitude
```

I tried using the second option but realized that it takes lot of time to process such huge file and at times, google service would time out. However, I think that if we can process our city data to derive address attribute for all the nodes using longitude and latitude then the resulting data set can provide lots of additional information. However, the decision is very subjective and is entirely dependent on the type of application. So the end user needs to decide between the tradeoff of cost and benefits associated with the approach.

Conclusion :

I came across several key issues with the data during my analysis and cleaning. Not all the nodes have the address field associated with it, which means that the data needs to be modified further to draw useful conclusions. Also the data ,initially had lots of other cities data as well, which I filtered out. There is inconsistency in terms of defining the cuisine for example, there are some cuisines with value burger. Clearly this data needs to be used very carefully in order to draw any useful inferences.