Project 2: OpenStreetMap Data Wrangling with MongoDB

Sanyam Chaudhary: Data Analyst November cohort

Map Area: Seattle, WA, United States:
http://www.openstreetmap.org/relation/237385#map=11/47.6076/-122.3424

Contents

Reasons for Choosing Seattle:

I have been in Seattle area for about 5 years and am intimately familiar with it. Hence I choose this area so that I can query for information and identify any incorrect information present in OpenStreet Maps data.

Problems encountered while wrangling the data.

Following section lists few problems I encountered while working with the given data set.

Challenges while preparing data for MongoDB:

Given the size of data, it required some efforts to import the data into MongoDB. I was consistently hitting the memory limits while importing the data, I was using the code we worked through in the class videos. Even after using cElementTree Python module, I was noticing consistent bump in memory while processing the xml data and after a while, the program would run out of memory.
Specifically, I was hitting the following issue: http://bugs.python.org/issue14762
Following the advice given in above link, I applied the .clear method after processing the xml element and voila the code worked! Below is the modified code which helped me covert the xml to well-formed JOSN. Also the Memory stayed same throughout i.e. no leaks.

Below is the code of shape_element function. As a result we now have a function which can parse as big an xml without hitting any memory limit.

```
def shape_element(element):
    node = {}
    address={}
    pos=[]
    created={}
    if element.tag == "node" or element.tag == "way" :
        attribs =element.attrib
        for key in attribs.keys():
            if key=='lat' or key=='lon':
                pos.append(float(attribs[key]))
            elif key in CREATED:
                created[key]=attribs[key]
            else:
                node[key]=attribs[key]
        for elem in element.iter("tag"):
                attribs = elem.attrib
                if(attribs.has_key("k")):
                    p = problemchars.match(attribs["k"])
                    if p is not None:
                        continue
                    l = lower.match(attribs["k"])
                    if l is not None:
                        node[attribs["k"]]=attribs["v"]
                        #print "lower==>"+ l.group()
                    lc = lower_colon.match(attribs["k"])
                    if lc is not None:
                         if lc.group()=="addr:street":
                            address['street']=audit_street_type(attribs["v"])
                         elif lc.group()[:lc.group().index(":")] == "addr":
                            keyname = lc.group()[lc.group().index(":")+1:]
                            val =attribs["v"]
                            address[keyname]=val
```

```
            #print keyname+" ==> "+ val
        elem.clear()        // Useful in avoiding the memory issue
    node["pos"]=pos
    node["created"]=created
    node["address"]=address
    node["type"]=element.tag
    element.clear()  // Useful in avoiding the memory issue
    return node
```

Street Names:

As suggested in the class assignments, I observed that street names are abbreviated. I used the same techniques as in class assignments to fix them and used the full name. For example

Moss St. -> Moss Street, Park Ln. -> Park Lane, Lake Washington Blvd: -> Lake Washington Boulevard.

Redundant field names:

Before extracting any information out of the data, it was imperative to look at data more closely and see what sort of data is available i.e. what different types of fields available are, is there any redundancy? Do I need to do more data cleaning before querying for specific? etc. Hence I wrote below script to query different document fields on all the docs.

```
var db = connect("sanyamc/test");
var res = db.Seattle.mapReduce(
        function(){emit(this._id,Object.keys(this));},
        function(_id,keys){return keys;},
        {out:"Seattle"+"_keys"} // results in a new collection Seattle_keys.

)
```

Then following query,

db.Seattle_keys.distinct("value").toString() resulted in following keys:

I am toString it and pasting below to reduce the space.

```
_id,address,amenity,created,id,name,pos,type,alt_name,created_by,source,website,ele,natural,odbl,
source_ref,building,operator,shop,cuisine,leisure,phone,wifi,dispensing,brand,opening_hours,ref,reli
gion,man_made,tourism,url,atm,description,aeroway,is_in,sport,entrance,landuse,designation,old_n
ame,notes,denomination,wikipedia,wheelchair,historic,start_date,official_name,note,wetland,iata,cit
y_served,closest_town,icao,access,faa,name_old,fee,internet_access,smoking,level,hairdresser,drive
_through,outdoor_seating,takeaway,food,office,barrier,email,church,doctor,healthcare,highw
ay,shelter,service,gym,short_name,unknown,vac,restaurant,city,state,storetype,kayak_rental,seattlei
mport,layer,trade,nails,int_name,fax,emergency,microbrewery,park,delivery,fuel,subcategory,double
checked,store,books,drive_in,old_alt_name,fixme,yelp,postal_code,pets,capacity,craft,breakfast
```

,clothes,college,medical,crop,todo,covered,residential,country,diplomatic,organization,min_age,height,repair,addr,social_facility,car_wash,reservation,add,park_ride,parking,bicycle,foot,motor_vehicle,door,information,organic,name_base,name_type,separated,from_address_left,from_address_right,reviewed,to_address_left,to_address_right,cycleway,lanes,maxspeed,oneway,reg_name,nat_name,surface,m,noname,bridge,old_ref,lit,sidewalk,horse,lcn,tunnel,area,smoothness,electrified,frequency,gauge,railway,voltage,attribution,loc_name,disused,power,substation,architect,construction,suite,condition,prop_description,rating,x_coordinate,y_coordinate,image,dog,floating,hov,maxheight,residents,automated,self_service,rest,recycling_type,toilets,diesel,ethanol,function,second_hand,maxstay,vacant,fenced,acres,owner,parks_id,perimeter,art,basketball,picnic,playground,restrooms,elevation,shape_area,shape_len,animal_shelter,gnis_feature_id,tomb,golf,supervised,webpage,social_facility_for,junction,ownership,place,laundry_service,anemity,boundary,shelter_type,fullname,gid,zipcode,hours,shooting

As you can see there is some cleaning required due to redundancy of info or typo occurred while adding data to openstreet. For example,

1. There were few docs with anemity instead of amenity.
2. Some fields had Zipcode or postal_code while most of them had address.postalCode.
3. Some docs had ToDo and Fixme fields specified for addresses.
4. Many-many docs didn't have address field.

However the values of Zipcode/Postal Code for the places I know did turn out to be fine.

After observing these discrepancies in field names, we can easily write scripts to clean data even further.

Updating/Renaming the field names:

As we can see from the table above that there are few field names with typo in it. Let's fix them.

Changing 'anemity' to 'amenity'

```
db.Seattle.update({'anemity':{$ne:null}},
                  {$rename:{'anemity':'amenity'}},
                   {multi:true})
```

Combining Zipcode, postal_code, into address.postcode

```
db.Seattle.update({'zipcode':{$ne:null}},
                  {$rename:{'zipcode':'address.postcode'}},
                   {multi:true})
db.Seattle.update({'postal_code':{$ne:null}},
                   {$rename:{'postal_code':'address.postcode'}},
                    {multi:true})
```

Updating the City Names

Seattle has lots of small suburbs around it so I got interested in knowing how much of the data we have for different cities. To start with I wanted to see different cities in Seattle data.

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

Here are top two entries:

```
{   "_id" : "Seattle",
     "count" : 202723
},
{    "_id" : "Kirkland",
     "count" : 42161
},
```

However on looking down the list, I noticed entries of the form:

```
 {   "_id" : "seattle", "count" : 7},
  { "_id" : "Seattle, WA", "count" : 2 },
 {  "_id" : "Seattle=1", "count" : 1 },
```

Also there were entries with Case difference and cities appended with State, for e.g.

```
{  "_id" : "Bellevue, WA", "count" : 31  },
{"_id" : "BELLEVUE","count" : 1 },
 {"_id" : "Bellevue", "count" : 401},
```

There were few typos in the city names too, for e.g. my residing city Kirkland was miss spelled as Kikaland etc.

Let's fix them:

```
db.Seattle.update({"address.city":{$in:["seattle","Seattle, WA","Seattle=1"]}},
                       {$set:{"address.city":"Seattle"}},
                       {multi:true})
db.Seattle.update({"address.city":{$in:["bellevue","Bellevue, WA","BELLEVUE"]}},
                       {$set:{"address.city":"Bellevue"}},
                       {multi:true})
db.Seattle.update({"address.city":{$in:["kikaland","Kirkland, WA","KIRKLAND","kirkland"]}},
                       {$set:{"address.city":"Kirkland"}},
                       {multi:true})
```

So looking at existing data confirms that there isn't any data validation in place while adding data to Openstreet and to analyze the data, we have to come up with iterative approach to clean the existing data. Given my familiarity with Seattle and surrounding cities it's relatively easy to do for this collection however if we were to clean data for all cities in Openstreet, that would pose some bigger and interesting challenge.

Data Overview:

Basic stats about the data and mongoDB queries

File Sizes:

Seattle.osm: 1.4 GB

Seattle.json: 1.5 GB

Number of documents:

> > db.Seattle.find().count()

6952489

Number of nodes:

> db.Seattle.find({type:"node"}).count()

6355942

Number of ways:

> db.Seattle.find({type:"way"}).count()

596547

Number of unique users:

> db.Seattle.distinct("created.user").length

2452

Top contributing user

```
> db.Seattle.aggregate([{"$group":{_id:"$created.user",count:{$sum:1}}},{$sort:{count:-1}},{$limit:1}])
{
	"result" : [
		{
			"_id" : "Glassman",
			"count" : 1117205
		}
```

```
        ],
        "ok" : 1
}
```

Additional data exploration using MongoDB queries

Seattle is famous for number of things which includes
a) Coffee, as Starbucks coffee started from Seattle. Let's find out if we have data around Starbucks.
b) Microsoft and Amazon Headquarters

a) Queries on Café shops

Number of starbucks coffee shops:

```
> db.Seattle.find({amenity:"cafe",name:/starbucks/i}).count()
```

211

Without using amenity:

```
> db.Seattle.find({name:/starbucks/i}).count()
```

227

Interesting, then I got interesting in knowing the amenity of shop named Starbucks which is not café.

```
> db.Seattle.findOne({amenity:{$ne:"cafe"},name:/starbucks/i}).amenity
```

Restaurant // AHA

Total Coffee Shops in Seattle:

```
> db.Seattle.find({amenity:"cafe"}).count()
```

816

Based on the above data, about 26% of the coffee shops are Starbucks.

b) Queries on commercial buildings.

Microsoft Corporation has a bunch of offices in Seattle area and so does Amazon. Let's find out how many buildings does each of these companies have.

```
> db.Seattle.find({"name":/microsoft/i,building:{$in:["commercial","office"]}}).count()
```

66

```
> db.Seattle.find({"name":/amazon/i,building:{$in:["commercial","office"]}}).count()
```

9

Conclusion:

Based on my familiarity with the Seattle area, I can see that data in maps is fairly complete. However there is a good amount of data cleaning that is required before we can account for all information while querying. Following are some suggested approaches:

a) Cleaning the sub-documents: In the earlier part of this document, I identified the field names present at the root level. There are multiple sub-documents and we can do similar exercise to identify any typo or incorrect format of the information in sub-docs and clean it further.

b) Checks while uploading the data: It appears that there is no data validation in place while uploading data to open street map. We can have some predefined parameters for any address which are a must, for e.g.

- Adding an address is must
- City names matching a list.
- Having a regex to check for the city names (ie should not include , wa)
- Converting the City names to a defined casing like Camel Casing etc.
- Validation for allowed zipcode range

c) Application of cleaned data: I can think of many applications of cleaned data. A well formatted and cleaned data can answer many questions. For e.g.

- Suppose as an individual I want to find different Italian restaurants in the city with their rating. I can do that fairly easily using mongodb. I can also have a web app which can query data from mongodb server to help me answer my questions.
- Other application is to perform statistics on the businesses/ data. For e.g. Number of Starbucks outdo the number of other coffee houses in Seattle. So while launching any new product/advertising, Starbucks can target Seattle as one of their big sample size of their customers.

d) Challenge of Data Validation: As with any map application, finding the validity of an address is a tough problem and would have to rely on the end users to fix that. Open street map can have another field for a data uploader suggesting the validity of information as perceived by others/users of openstreet.

References:

1. Python documentation for Python language.

https://docs.python.org/2/library/functions.html

2. MongoDB Aggregation and Map reduce concepts:

http://docs.mongodb.org/manual/core/aggregation/

http://docs.mongodb.org/manual/core/map-reduce/

3. StackOverflow and following link for help with XML parsing issue described above:

http://bugs.python.org/issue14762

4. Udacity Course on MongoDB

https://www.udacity.com/course/viewer#!/c-ud032-nd/l-768058569/e-844308648/m-845648636