OpenStreetMap Project
Data Wrangling with MongoDB

Map Area: Los Angeles, CA, United States

Problems Encountered in the Map

After downloading the entire data for Los Angeles and parse it iteratively to collect samples of size 100, I noticed three main problems with the data:

- Inconsistency in full address ("South Freemont Ave", "Clearglen Avenue"). Some even have uncomplete full address.
- Inconsistent house numbers ("2475 Adriatic Ave.", "704 Ste 4", "150 A")
- Inconsistent street names ("Ramona Street", "Don Julian Rd","S. Francisca Avenue")
- Incorrect zip code (Los Angeles zip codes range from 90001 to 91607, but some zip codes are outside this range)

Inconsistency in Full Address:

Once the data was imported to MongoDB, some queries revealed that a lot of nodes do not have a full address. Nodes that do have full address often represent them in inconsistent manner. I realize that a node can either be an actual entity with a valid address (building, house, etc.) or one without a valid address (public toilet, stop sign, etc.). If a node is a valid building, it makes sense that such node will have the value house number at the least. In fact, there is a lot of nodes that either missing street address, zip code, city, country, state or all of the above. Therefore, there is a need to update all these fields so that if a node is representing a building, it will be defined by these standard parameters. Each node will also include a full address in order to make querying much easier for users. For example: I added the following full address to one of the nodes: "2426, Gum Tree Lane, San Diego, California, 92028, United States of America"

Inconsistent Street Names:

Inconsistency in street names pose another problem. Since street types are not consistent, users will encounter difficulties when doing queries based on street names. For example, a user might search for information about a place by typing in its address: "2475 Adriatic Ave", but the actual street address stored is "2475 Adriatic Avenue". Thus, the query might return no result. Most of the street types are in abbreviated form, and since there is not enough information to fully translate the abbreviated street types, it is more efficient if all the explicitly spelled out street types are converted into abbreviated form. Thus, I downloaded the street suffix data from () and use it to create a dictionary for street types mapping. By programmatically look up the mapping dictionary, all street types are converted into abbreviated form.

Inconsistent House Numbers:

House numbers pose another problem. A few house numbers don't follow a particular pattern. Some house numbers contain street address ("2475 Adriatic Ave."), some contain abbreviated word for "Suite" ("324 Ste 3"). This inconsistency creates problems when users want to query a particular house numbers. Therefore, Using regular expression, I was able to find problematic house numbers, extract the numbers from the street address, and explicitly spells out "Suite" if the address is a suite address ("324 Ste 3" became "324 Suite 3")

Zip Codes:

Zip codes are another important issue that needs to be fixed. Zip codes in the data are Unicode-encoded and represented in inconsistent form. Some zip codes contain only numbers ("90032") while other contains words ("CA 90002" or "Ca 90024"). Since zip codes are Unicode strings, it's hard to query for a certain combination of zip codes. For example, if we only want the addresses in Los Angeles city, we can search for zip codes within the range 90001 – 91607. But since zip codes are Unicode string, this type of query is impossible. Therefore, I used regular expression to get the problematic zip codes, change them to normal numbers ("CA 90001" became "90001"), and then convert all zip codes to integer numbers.

Incorrect Zip Codes:

The zip codes obtained from Los Angeles Open Street Map data also includes zip codes from suburban area around Los Angeles. Los Angeles zip codes range from 90001 to 91607, but a lot of zip codes fall outside this range. Postal codes are grouped together using this aggregator:

```
# Sort postcodes by count, descending

db.osm.node.aggregate([{"$match":{"tag.addr:postcode":{"$exists":Tru
e}}},
                       {"$group":{"_id":"$tag.addr:postcode",
                        "count":{"$sum":1}}},
                       {"$sort":{"count":-1}},
                       {"$limit":3}])
```

Here are the top three results, beginning with the highest count:

```
{u'count': 12644, u'_id': 92630}
{u'count': 12398, u'_id': 92028}
```

```
{u'count': 7045, u'_id': 92860}
```

After some research, I was able to find out that the zip 92630 corresponds to Lake Forest, CA, which is about 56 miles from Los Angeles. Thus, I believed that this data should be referred to as Los Angeles Metropolitan Area, since it also includes surrounding cities.

Data Overview

This section contains basic statistics about the dataset and the MongoDB queries used to gather them

File size:

los-angeles_california.osm……1.13 GB

Number of documents

Since I separate the data into three collections: “node”, “relation”, and “way”, the total count of documents will be the sum of the counts of documents in these two collections:

```
db.osm.node.find().count()……256914
db.osm.way.find().count()……..569165
db.osm.relation.find().count()……..6665
```

Number of nodes

```
db.osm.node.find().count()……256914
```

Number of ways

```
db.osm.way.find().count()……..569165
```

Number of relations:

```
db.osm.relation.find().count()……..6665
```

Number of unique users

len(db.osm.node.distinct(“created.user”))……414

len(db.osm.way.distinct(“created.user”))……..983

len(db.osm.node.distinct(“created.user”))……354

Top 1 contributing user:

```
db.osm.node.aggregate([{“$match”:{“created.user”:{“$exists”:True}}},{"$
group":{"_id":"$created.user", "count":{"$sum":1}}}, sort":{"count":-
1}}, {"$limit":1}])
{u’id’: u’SJFriedl’, u’count’: 27770}

db.osm.way.aggregate([{“$match”:{“created.user”:{“$exists”:True}}},{"$g
roup":{"_id":"$created.user", "count":{"$sum":1}}}, sort":{"count":1}},
{"$limit":1}])
{u’id’: u’Jon Schleuss’, u’count’: 10291}

db.osm.relation.aggregate([{“$match”:{“created.user”:{“$exists”:True}}}
,{"$group":{"_id":"$created.user",
"count":{"$sum":1}}}, sort":{"count":1}}, {"$limit":1}])
{u’id’: u’AM909’, u’count’: 2315}
```

Number of users appearing only once:

```
db.osm.node.aggregate([{"$match":{"created.user":{"$exists":True}}},
{"$group":{"_id":"$created.user", "count":{"$sum":1}}},
{"$group":{"_id":"$count", "num_users":{"$sum":1}}},
{"$sort":{"_id":1}}, {"$limit":1}])
{u’id’: 1, u’num_users’: 255}

db.osm.way.aggregate([{"$match":{"created.user":{"$exists":True}}},
{"$group":{"_id":"$created.user", "count":{"$sum":1}}},
{"$group":{"_id":"$count", "num_users":{"$sum":1}}},
{"$sort":{"_id":1}}, {"$limit":1}])
{u’id’: 1, u’num_users’: 321}

db.osm.relation.aggregate([{"$match":{"created.user":{"$exists":True}}}
,
{"$group":{"_id":"$created.user", "count":{"$sum":1}}},
{"$group":{"_id":"$count", "num_users":{"$sum":1}}},
{"$sort":{"_id":1}}, {"$limit":1}])
{u’id’: 1, u’num_users’: 140}
```

Additional Ideas

I realize that this data can be used in many useful ways. One of the way is to combine this data with Los Angeles Subway Turnstile data. In other classes at Udacity, I have learnt how to utilize subway data and machine learning to make predictions about subway ridership. By using the turnstile data, we can approximately predict the crowdedness of subway on a particular days, and suggest different routes based on the Los Angeles Open Street Map data. The Open Street Map data can also be used to optimize user's experience by calculating the most cost-effective method of travelling. For example, it can be used to calculate the shortest road that have high speed limit (to reduce time), which also have low-cost foods and accommodations along the way. Thus, by combining with the Los Angeles Subway Turnstile data, an application can be created to help subway riders take the most out of their time and money.

Such implementation requires the data to be improved in certain ways. One of them is the inclusion of the average costs of meals and accommodations. If possible, service ratings can also be included in the dataset as well. Such improvement may increase the size of data to be processed, but this increase can be offset by standardize the data structure for each nodes. For example, in the following node of "Bravo Avo" restaurant, we can replace the smoking tag (which is not particularly important because most restaurants don't allow smoking) by average price tag:

```
{u'_id': ObjectId('5625e30916595d0a00d40c15'),
 u'changeset': u'32273176',
 u'id': u'3381783301',
 u'lat': u'33.6811368',
 u'lon': u'-117.6657508',
 u'tag': {u'amenity': u'restaurant',
          u'cuisine': u'mediterranean',
          u'name': u'Bravo Avo',
          u'smoking': u'no',
          u'takeaway': u'yes',
          u'website': u'http://bravoavo.com'},
 u'timestamp': u'2015-06-28T22:25:33Z',
 u'type': u'node',
 u'uid': u'2709708',
 u'user': u'SJFriedl',
 u'version': u'3'}
```

By optimizing the data represented in each particular node, Open Street Map data can be used for various census research and analysis, more than just for looking up street data.

Additional data exploration using MongoDB queries

```
# Top 10 appearing amenities
```

```
db.osm.node.aggregate([{"$match":{"tag.amenity":{"$exists":True}}},
                       {"$group":{"_id":"$tag.amenity",
                                  "count":{"$sum":1}}},
                       {"$sort":{"count":-1}},
                       {"$limit":10}])
```

```
{u'_id': u'place_of_worship', u'count': 3793}
{u'_id': u'school', u'count': 3128}
{u'_id': u'restaurant', u'count': 1786}
{u'_id': u'fast_food', u'count': 1258}
{u'_id': u'fuel', u'count': 742}
{u'_id': u'cafe', u'count': 594}
{u'_id': u'toilets', u'count': 504}
{u'_id': u'parking', u'count': 493}
{u'_id': u'fountain', u'count': 493}
{u'_id': u'drinking_water', u'count': 473}
```

Most popular cuisines

```
db.osm.node.aggregate([{"$match":{"tag.amenity":{"$exists":1},
                        "tag.cuisine":{"$exists":1},
                        "tag.amenity":"restaurant"}},
                       {"$group":{"_id":"$tag.cuisine",
                                  "count":{"$sum":1}}},
                       {"$sort":{"count":-1}},
                       {"$limit":2}])
```

```
{u'_id': u'american', u'count': 159}
{u'_id': u'mexican', u'count': 151}
```

Most popular hospital

```
db.osm.node.aggregate([{"$match":{"tag.amenity":{"$exists":True},
                        "tag.name":{"$exists":True},
                        "tag.amenity":"hospital"}},
                       {"$group":{"_id":"$tag.name",
                                  "count":{"$sum":1}}},
                       {"$sort":{"count":-1}},
                       {"$limit":2}])
```

```
{u'_id': u'Samimi Orthopedic Group', u'count': 2}
{u'_id': u'First Aid', u'count': 2}
```

Most popular place of worship:

```
db.osm.node.aggregate([{"$match":{"tag.amenity":{"$exists":True},
                        "tag.name":{"$exists":True},
```

```
                                "tag.amenity":"place_of_worship"}},
                        {"$group":{"_id":"$tag.name",
                                "count":{"$sum":1}}},
                        {"$sort":{"count":-1}},
                        {"$limit":2}])
```

```
{u'_id': u'The Church of Jesus Christ of Latter-day Saints', u'count':
77}
{u'_id': u'Church of Christ', u'count': 67}
```

Conclusion

The Open Street Map data is already pretty standardized and clean when I first take a look at the data, yet there is still room for improvement to make the data much more efficient and effective for querying. By using the osm.py I provided, every node that has a house number will include all the important components of a full address (house number, street, city, state, post code, country), as well as a complete spelled out full address for bulk query. Also, all the street names and house number will be standardized and every node will follow the same data structure with “address” and “created” fields. I believe that by implementing robust data processor, it would be possible to input a great amount of cleaned data to OpenStreetMap.org.