Open Street Map Project

Ye Peng Li

1. Problems Encountered in the Map

After initially downloading a small sample size of the Houston area and running it against a provisional data.py file, I noticed three main problems with the data, which I will discuss in the following order:

Over-abbreviated street names (“'E FM 528 Rd'”)

Inconsistent postal or incorrect postal code ('TX 77009','77007-2113', '77007','7-')

Incorrect phone number ('phone=+1 713 235 8800')

Phone number without country code.

Over-abbreviated Street Names

Once the data was imported to MongoDB, some basic querying revealed street name abbreviations and postal code inconsistencies. I updated all substrings in problematic address strings, such that “W Clayton St'” becomes “West Clayton Street”.

Postal Codes

Postal code strings posed a different sort of problem, forcing a decision to strip all leading and trailing characters before and after the main 5- digit zip code. This effectually dropped all leading state characters (as in “TX77494”) and 4-digit zip code extensions following a hyphen (“77007-2121”). This 5-digit constriction benefits MongoDB aggregation calls on postal codes. I update this kind of postal code to five digits through mapping function.

Regardless, after standardizing inconsistent postal codes, some altogether “incorrect” (or perhaps misplaced?) postal codes surfaced when check the audit report: Houston's postal code is ranging in 77001 to 77297.But I find a lot of postal code like '77339','77459'.Part 5 "postal_code_sort" and "city_sort" results confirmed my suspicion that this metro extract would perhaps be more aptly named “Metrolina” or the “Houston Metropolitan Area” for its inclusion of surrounding cities in the sprawl.

So, these postal codes aren’t “incorrect,” but simply unexpected.

```
postal_code_sort =
db.Houston109.aggregate([{"$match":{"address.postcode":{"$exists":1}}},

                                        {"$group":{"_id":"$address.postcode",

                                                   "count":{"$sum":1}}},

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

{u'count': 481, u'_id': 77096}{u'count': 253, u'_id': 77449}

```
city_sort = db.Houston109.aggregate([{"$match":{"address.city":{"$exists":1}}},
{"$group":{"_id":"$address.city", "count":{"$sum":1}}}, {"$sort":{"count":1}}])
```

{u'count': 1, u'_id': u'Laks Jackson'}{u'count': 1, u'_id': u'Plantersville'}

2. Overview of the data

Houston_texas.osm:	710MB (unpressed)
Houston_texas.osm.json	624MB (unpressed)

```
#Number of ddocuments,nodes and ways

print "The total number of documents:
{0}".format(db.Houston111.find().count())

print "The total number of nodes:
{0}".format(db.Houston111.find({"type" : "node"}).count())

print "The total number of ways:
{0}".format(db.Houston111.find({"type" : "way"}).count())
```

```
The total number of documents: 3209780
The total number of nodes: 2865564
The total number of ways: 344195
```

#Number of Unique users in Houston, TX

```
number_of_unique_users = db.Houston111.aggregate([{"$group" : {"_id" : "$created.user","count" : {"$sum" : 1}}}])

user_count = 0

for doc in number_of_unique_users: user_count += 1

print user_count
```

```
Number of unique users              1283
```

 # Numbers of distinct amenities

```
distinct_amenities = db.Houston111.distinct("amenity")

number_of_amenities = len(distinct_amenities)

print "The number of distinct amenities: {0}".format(len(distinct_amenities))
```

```
The number of distinct amenities: 98
```

 # Numbers of schools

```
number_of_schools = int(db.Houston111.find({"amenity" : "school"}).count())

print "The total number of schools: {0}".format(number_of_schools)
```

```
The total number of schools: 1649
```

 # Numbers of cafes

```
number_of_cafes = int(db.Houston111.find({"amenity" : "cafe"}).count())

print "The number of cafes: {0}".format(number_of_cafes)
```

```
The number of cafes: 126
```

```
# Number of users appearing only once (having 1 post)

one_post_user =
db.Houston111.aggregate([{"$group":{"_id":"$created.user",
"count":{"$sum":1}}}, {"$group":{"_id":"$count",
"num_users":{"$sum":1}}}, {"$sort":{"_id":1}},
{"$limit":1}])

for user in one_post_user: print user
```

```
{u'num_users': 248, u'_id': 1}
```

3. Getting stats from dataset

Contributor statistics and gamification suggestion

The contributions of users seems incredibly skewed, possibly due to automated versus manual map editing (the word “bot” appears in some usernames). Here are some user percentage statistics:

a. Top user contribution percentage (“woodpeck_fixbot”) - 18.42%.

b. Combined top 2 users' contribution (“jumbanho” and “woodpeck_fixbot”) - 35.49%.

c. Combined Top 5 users contribution - 61.01%.

d. Combined number of users making up only 0.0077% of posts - 248 (about 20% of all users).

Thinking about these user percentages, I’m reminded of “gamification” as a motivating force for contribution. In the context of the OpenStreetMap, if user data were more prominently displayed, perhaps others would take an initiative in submitting more edits to the map. And, if everyone sees that only a handful of power users are creating more than 60% a of given map, that might spur the creation of more efficient bots, especially if certain gamification elements were present, such as rewards, badges, or a leaderboard.

4. Additional data exploration using MongoDB queries

#Top 5 contributory users

```
top_5_users = db.Houston111.aggregate([

                {"$group" : {"_id" : "$created.user",

                "count" : {"$sum" : 1}}},

                {"$sort" : {"count" : -1}},

                {"$limit" : 5 }])

for doc in top_5_users: print doc
```

```
{u'count': 591156, u'_id': u'woodpeck_fixbot'}
{u'count': 547958, u'_id': u'TexasNHD'}
{u'count': 425099, u'_id': u'afdreher'}
{u'count': 199984, u'_id': u'scottyc'}
{u'count': 193936, u'_id': u'cammace'}
```

#The five most popular amenities in Houston, TX

```
five_most_popular_amenities =
db.Houston111.aggregate([{"$match" : {"amenity" : {"$exists" :
1}}},{"$group" : {"_id" : "$amenity","count" : {"$sum" :
1}}},{"$sort" : {"count" : -1}},{"$limit" : 5}])

for doc in five_most_popular_amenities: print doc
```

```
{u'count': 3496, u'_id': u'parking'}
{u'count': 2452, u'_id': u'place_of_worship'}
{u'count': 1649, u'_id': u'school'}
{u'count': 904, u'_id': u'fast_food'}
{u'count': 828, u'_id': u'restaurant'}
```

#Biggest religion

```
biggest_religion =
db.Houston111.aggregate([{"$match":{"amenity":{"$exists":1},
"amenity":"place_of_worship"}},
{"$group":{"_id":"$religion", "count":{"$sum":1}}},
{"$sort":{"count":-1}}, {"$limit":1}])
```

```
for b in biggest_religion:print b
```

```
{u'count': 2376, u'_id': u'christian'}
```

 # Most popular cuisines

```
most_pop_cuisine =
db.Houston111.aggregate([{"$match":{"amenity":{"$exists":1},
"amenity":"restaurant"}},{"$group":{"_id":"$cuisine",
"count":{"$sum":1}}},{"$sort":{"count":-1}}, {"$limit":2}])
```

```
for m in most_pop_cuisine:print m
```

```
{u'count': 399, u'_id': None}
{u'count': 82, u'_id': u'mexican'}
```

5.Conclusion

After this review of the data it's obvious that the Houston metro area is incomplete, though I believe it has been well cleaned for the purposes of this exercise. It interests me to notice a fair amount of GPS data makes it into OpenStreetMap.org on account of users' efforts, whether by scripting a map editing bot or otherwise. With a rough GPS data processor in place and working together with a more robust data processor similar to data.py I think it would be possible to input a great amount of cleaned data to OpenStreetMap.org.

Another interesting problem is that the query result about most popular cuisine are "None"(399) and "Mexican"(82).If there is more information collected about the unknown type of cuisine it will be better for people to know more about the favorite food of Houston residents. Especially for people who want to invest a restaurant such information is very important. On the other hand adding more detailed information to the unknown type of cuisine can make this part of data easier for other people to querying. It also helps to add a great amount of cleaned data to OpenStreetMap.org.

What's more, the positions that are saved for each "node" document can be used to look at not only how popular a given amenity/cuisine type is, but how densely populated certain areas of the city are by those amenities. With that type of information I could compare two cities and look at a potential proxy for obesity by looking at the densities of fast food establishments between the two cities.