Udacity Project 3
OpenStreetMap Project
Data Wrangling with MongoDB

Wonjun Lee

Overview

Mclean Virginia area data is extracted from the OpenStreetMap website.
I created 3 python scripts: “finalProject_osm_to_json.py”, “finalProject_dbinsert.py”, and “finalProject_aggregate.py”.

Data Overview

Problems Encountered

During the process of parsing the source of the data, many are redundant. For example, “Bing” is listed in a lot of different ways: “BING”, “bing”, “bing imagery”, etc. I parsed it manually since there are only few number of sources listed in the data. About 1.2% of the data contain the source. Below is the dictionary I used to parse the source of the data.

```
def fix_source(db):
    """ Fix sources """
    fix_sources = {"Bing"  : ['Bing; knowledge; logic','bing imagery,_data, field papers,on-site','bing imagery,_data,field papers,on-site',"binng", "BING", "bing", "bing imagery", "Bing imagery", "bing imagery, _data,firld papers,on-site", 'bing imagery, _data, field papers, on-site', "biung", "Bing, site visit"],
                "Yahoo" : ["Yahoo imagery", "yahoo"],
                "site visit" : ["Site visit", "imagery", "site survey", "GPS, site visit"],
                "ground truth" : ["ground truthing"],
                "fairfaxtrails.org" : ['http://www.fairfaxtrails.org', 'http://www.fairfaxtrails.org/pimmit/110707Legal_brochures_updown.pdf'],
                "Fairfax County GIS" : ['http://www.fairfaxcounty.gov/library/branches/dm/','Fairfax County Free GIS data','www.fairfaxcounty.gov > Tax Records property map 0602010037','Fairfax County GIS (http://www.fairfaxcounty.gov/maps/metadata.htm)','county_import_v0.1_20080508235459'],
                "knowledge" : ['from walking it','ground truth','I work there','local knowledge','In-person Source, ate there'],
                "survey" : ["ground survey"],
                "Tiger" : ['TIGER/Line 2008 Place Shapefiles (http://www.census.gov/geo/www/tiger/)', "Tiger2008 by DaleP 2009-02-28"],
                "DCGIS" : ['DCGIS; NPS','DCGIS; NPS; Park Service Map; USGS NM',"dcgis"]
```

Beside the redundancy in the sources, there are some sources I don’t know so I had to google them to research about them.

I looked at the map data and found that there are formatting problems in street names, postcodes and phone numbers.

Street names are easy to fix because there are only few abbreviated names so I created a dictionary.

```
def parse_street(street):
```

```
    dic = {"Ct": "Court",
           "Blvd": "Boulevard",
           "Ave": "Avenue",
           "E": "East",
           "Rd": "Road",
           "Pl": "Place"
          }
    if street.split(" ")[-1] in dic.keys():
        street = street.replace(street.split(" ")[-1], dic[street.split(" ")[-1]])
        return street
    else:
        return street
```

The only odd postcodes contain “-“ after 5 digits. I took out any digits after “-“.

```
def parse_postcode(postcode):
    return postcode.split("-")[0]
```

Some odd phone numbers contain country code “+1”. And there are some formats that have to be fixed such as “(###) ### - ####”, “### - ### - ####”, “###.###.####”, etc. I used regex and a sub method to take out all the unnecessary symbols.

```
phonechars = re.compile(r'[().\-\. ]')

def parse_phone(phone):
    if phone[0] == "+":
        phone = phone[2:]
    return phonechars.sub("",phone)
```

The most difficult problem was during the “Additional Ideas” section. I tried to calculate the number of houses around at each metro station. I found 9 metro stations from the database and 3 of them are nodes and 6 of them are ways. If they are nodes then I can extract the position data very easily but when they are ways then I need to think of a different way to find the position from the database. Each way contains more than 1 node in “node_refs” so I decided to use the first node. Now I have 1 node for each way. I extracted the position data for each node from the mongoDB database.

When I tried to find the number of house within the range I designed, I also needed the position data for the houses. Unfortunately houses are ways so I had to do the same thing as I did for the ways of metros. At first I tried to find the position data for each house node just as I did when I found the position for each node of metro stations, and it took about 10 minutes. There were only 6 metro stations, but there were more than thousand houses. I realized that aggregating the data for each house was very inefficient. I read through the mongoDB document to figure out more efficient way of doing it and I found I can use “$in” to avoid aggregating the database for each house. After this modification, the script took less than 1 minute to run.

File Sizes

map 77,783 KB
map.json 115,040 KB

Convert osm to json

I only extracted node and way data from the map osm file.
This is the format of node data and way data written in the json file.

```
node = {
  "id": None,
  "visible": None,
  "type": "node",
  "railway": None,
  "amenity": None,
  "name": None,
  "pos": {
    "lat": None,
    "lon": None
    },
  "created" : {
    "changeset": None,
    "user": None,
    "version": None,
    "uid": None,
    "timestamp": None,
    "source" : None
    }
  }
```

```
way = {
  "id" : None,
  "type": "way",
  "address":{},
  "railway": None,
  "name": None,
  "building" : None,
  "created" : {
    "changeset": None,
    "user": None,
    "version": None,
    "uid": None,
    "timestamp": None,
    "source" : None
  }
}
```

Insert the json file to mongoDB

```
In: db.map.find_one()
Out: {u'amenity': None, u'name': None, u'created': {u'changeset': u'19557774', u'uid': u'1677159',
u'timestamp': u'2013-12-20T22:10:17Z', u'source': None, u'version': u'3', u'user': u'Jason Gottshall'},
u'pos': {u'lat': 38.869535, u'lon': -77.1495846}, u'visible': None, u'railway': None, u'_id':
ObjectId('572ed501c7f1e9250cfe1570'), u'type': u'node', u'id': u'246574'}
In: db.map.count()
Out: 398663
```

Data Analysis

 # Total Number of Records

```
In: db.map.count()
Out: 398663
```

 # Total Number of Nodes

```
In: db.map.find({“type”:”node”}).count()
Out: 353600
```

 # Total Number of Ways

```
In: db.map.find({“type”:”way”}).count()
Out: 45063
```

 # Total Number of Unique Users

```
In: len(db.map.distinct(“created.user”))
```

Out: 531

 # Total Number of Unique Sources
In: db.map.distinct(“created.source”)
Out: 61

 # Top Contributing Users
In: top_user = db.map.aggregate([{"$group":{"_id":"$created.user", "count":{"$sum":1}}},
{"$sort":{"count":-1}}, {"$limit":5}])
In: for doc in top_user:
    print doc
out:
{u'count': 133558, u'_id': u'ingalls'}
{u'count': 40190, u'_id': u'woodpeck_fixbot'}
{u'count': 38561, u'_id': u'Your Village Maps'}
{u'count': 26405, u'_id': u'shoe'}
{u'count': 20858, u'_id': u'kriscarle'}

 # Top Sources
In: top_source = db.map.aggregate([{"$group":{"_id":"$created.source", "count":{"$sum":1}}},
{"$sort":{"count":-1}}, {"$limit":5}])
in: for doc in top_source:
    print doc
out:
{u'count': 394065, u'_id': None}
{u'count': 3385, u'_id': u'Bing'}
{u'count': 420, u'_id': u'Fairfax County GIS'}
{u'count': 290, u'_id': u'Yahoo'}
{u'count': 106, u'_id': u'survey'}

 # Number of One Time users
In: one_time_users = db.map.aggregate([{
                    "$group": {
                      "_id": "$created.user",
                      "count": { "$sum" : 1}
                      }
                    },
                    {
                    "$match": {
                      "count" : 1
                      }
                    }
                  ])
  count = 0

```
   one_time_users_list = []
   for user in one_time_users:
      one_time_users_list.append(user)
      count += 1
   count
out: 108

# Top Amenities
In: num_metros = db.map.aggregate([
         {
         "$match": {"amenity": {"$ne": None}}
         },
         {
         "$group": {"_id": "$amenity", "count": {"$sum": 1}}
         },
         {
         "$sort" : {"count": -1}
         },
         {
         "$limit": 5
         }
      ])
   for i in num_metros:
      print i
out:
{u'count': 173, u'_id': u'restaurant'}
{u'count': 101, u'_id': u'place_of_worship'}
{u'count': 79, u'_id': u'school'}
{u'count': 57, u'_id': u'fuel'}
{u'count': 49, u'_id': u'fast_food'}

# Number of types of Amenities
In:    num_metros = db.map.aggregate([
         {
         "$match": {"amenity": {"$ne": None}}
         },
         {
         "$group": {"_id": "$amenity"}
         },
         {
         "$group": {"_id": None, "count": {"$sum": 1}}
         }
      ])
   for i in num_metros:
```

```
    print "\nNumber of Amenities:", i["count"]
out: 59

# Number of amenities exists in the data
In:   num_metros = db.map.aggregate([
        {
        "$match": {"amenity": {"$ne": None}}
        },
        {
        "$group": {"_id": None, "count": {"$sum": 1}}
        }
      ])
   for i in num_metros:
      print i["count"]
out: 872

# Number of Schools
In:   num_metros = db.map.aggregate([
        {
        "$match": {"amenity": "school"}
        },
        {
        "$group": {"_id": None, "count":{"$sum":1}}
        }
      ])
   for i in num_metros:
      print i["count"]
out: 79

#Number of Buildings
In:   num_metros = db.map.aggregate([
        {
        "$match": {"building": {"$ne": None}}
        },
        {
        "$group": {"_id": None, "count":{"$sum": 1}}
        }
      ])
   for i in num_metros:
      print "\nNumber of Buildings:", i["count"]
out: Number of Buildings: 27684
```

Additional Statistics

 # Percentage of Top Source – "None": 98.8466449106 %

 # Percentage of 2nd Top Srouce – "Bing": 0.849088076897 %

 # Percentage of Top User – "ingalls": 33.5014786925 %

 # Percentage of Top Amenity – "restaurant": 19.8394495413 %

Additional Ideas

I am living in this area and I know a lot of people who try to find the house near the metro stations. So I decided to find the metro station with the largest number houses around it.

First I found the number of metros in my map collection.

In:

```
# Getting metro station data
""" Number of Metros """
metros = db.map.aggregate([
        {
        "$match": {"railway": "station"}
        }
        # {
        # "$project": {"railway": "$railway",
        #             "name" : "$name",
        #             "type" : "$type"}
        # }
    ])
print "\nMetros"
metro_lists = []
for i in metros:
    print i["name"], "-", i["type"]
    if i["type"] == "node":
        print "position:", i["pos"]
    metro_lists.append(i)
```

Out:

```
Metros
East Falls Church - node
position: {u'lat': 38.8859763, u'lon': -77.1568243}
Vienna/Fairfax-GMU - node
```

```
position: {u'lat': 38.8776013, u'lon': -77.2722884}
West Falls Church Metro - node
position: {u'lat': 38.9007928, u'lon': -77.1889651}
Spring Hill - way
Greensboro - way
Tysons Corner - way
McLean - way
West Falls Church-VT/UVA - way
Dunn Loring-Merrifield - way
```

Here the problem occurs. Way information doesn't have a position data and it only has nodes. According to the result above, 6 out of 9 are ways.

Each way has "node-refs" information and I am going to take the first node from each way data and create a new dictionary that links metro names to nodes.

In:

```
print "\nFind the first node from way information"
way_nodes = {}
for i in metro_lists:
    if i["type"] == "way":
        way_nodes[i["name"]] = i["node_refs"]
pprint.pprint(way_nodes)
```

Out:

```
Find the first node from way information
{u'Dunn Loring-Merrifield': u'2363986739',
 u'Greensboro': u'2362666881',
 u'McLean': u'2362684874',
 u'Spring Hill': u'2362647097',
 u'Tysons Corner': u'2362670319',
 u'West Falls Church-VT/UVA': u'2363747447'}
```

Then I aggregate the map collection from the MongoDB find the position information from the node data.

In:

```
print "\nFind the position of each node of metros "
nodes_pos = {}
for node in way_nodes.values():
    db_way_nodes = db.map.aggregate([
        {
        "$match": {"id": node}
        },
```

```
            {
            "$project": {"node": "$id",
                    "pos" : "$pos"}
            }
        ])
    for i in db_way_nodes:
        nodes_pos[i["node"]] = i["pos"]
```

Finally, I create a new dictionary that connects metro names to position data.

In:

```
way_pos = {}
for metro in way_nodes:
    way_pos[metro] = nodes_pos[way_nodes[metro]]

way_pos["East Falls Church"] = {'lat': 38.8859763, 'lon': -77.1568243}
way_pos["Vienna/Fairfax-GMU"] = {'lat': 38.8776013, 'lon': -77.2722884}

print "Way to position:"
pprint.pprint(way_pos)
```

Out:

```
Create metros to positions
Way to position:
{u'Dunn Loring-Merrifield': {u'lat': 38.8832183, u'lon': -77.2288656},
 'East Falls Church': {'lat': 38.8859763, 'lon': -77.1568243},
 u'Greensboro': {u'lat': 38.9219619, u'lon': -77.2347193},
 u'McLean': {u'lat': 38.9248036, u'lon': -77.2093675},
 u'Spring Hill': {u'lat': 38.9285203, u'lon': -77.2413415},
 u'Tysons Corner': {u'lat': 38.9206611, u'lon': -77.2235898},
 'Vienna/Fairfax-GMU': {'lat': 38.8776013, 'lon': -77.2722884},
 u'West Falls Church-VT/UVA': {u'lat': 38.9012072, u'lon': -77.188819}}
```

I took out “West Falls Church Metro” because it is same as “West Falls Church-VT/UVA”. So I have total 8 metros in my map data.

I need to have information about houses in the map data. According to the dataset, there are many types of buildings and I need to see which types are related to the residential building.

In:

```
print "\nFind the type of buildings"
# Find residential type of buildings
num_metros = db.map.aggregate([
        {
        "$match": {"building": {"$ne": None}}
```

```
        },
        {
        "$group": {"_id": "$building", "count": {"$sum": 1}}
        },
        {
        "$sort": {"count": -1}
        }
    ])
  for i in num_metros:
    pprint.pprint(i)
```

Out:

```
Find the type of buildings
{u'_id': u'yes', u'count': 21572}
{u'_id': u'detached', u'count': 2372}
{u'_id': u'residential', u'count': 1371}
{u'_id': u'house', u'count': 1267}
{u'_id': u'apartments', u'count': 339}
{u'_id': u'garage', u'count': 143}
{u'_id': u'office', u'count': 136}
{u'_id': u'Townhouse', u'count': 118}
{u'_id': u'retail', u'count': 97}
{u'_id': u'commercial', u'count': 59}
{u'_id': u'roof', u'count': 49}
{u'_id': u'terrace', u'count': 39}
{u'_id': u'school', u'count': 36}
{u'_id': u'public', u'count': 17}
{u'_id': u'industrial', u'count': 17}
{u'_id': u'shed', u'count': 16}
{u'_id': u'church', u'count': 13}
{u'_id': u'no', u'count': 7}
{u'_id': u'hotel', u'count': 4}
{u'_id': u'manufacture', u'count': 2}
{u'_id': u'walkway', u'count': 2}
{u'_id': u'Pumping_Station', u'count': 1}
{u'_id': u'hospital', u'count': 1}
{u'_id': u'canopy', u'count': 1}
{u'_id': u'barn', u'count': 1}
{u'_id': u'bleachers', u'count': 1}
{u'_id': u'university', u'count': 1}
{u'_id': u'warehouse', u'count': 1}
{u'_id': u'parking_garage', u'count': 1}
```

Among the types of buildings above, “apartments”, “residential”, “house”, and “Townhouse” are residential buildings. I filter these residential buildings and extract the node information.

In:

```
num_metros = db.map.aggregate([
      {
      "$match": {
            "$or":
            [
            {"building": "apartments"},
            {"building": "residential"},
            {"building": "house"},
            {"building": "Townhouse"}
            ]
        }
      },
      {
      "$project": {"node": "$node_refs"}
      }
   ])

array_building_nodes = []
for i in num_metros:
   array_building_nodes.append(i["node"])
```

In:

```
db_building_nodes_pos = db.map.aggregate([
      {
      "$match": {
         "type": "node",
         "id": {"$in" : array_building_nodes}
         }
      }
   ])
building_pos = []
for i in db_building_nodes_pos:
   lat = i["pos"]["lat"]
   lon = i["pos"]["lon"]
   building_pos.append([lat,lon])

print "\nLength of building:", len(building_pos)
```

Out:

```
Length of building: 3089
```

There are 3089 houses in the dataset. Now I have all the information I need to calculate the number of houses near each metro. I am going to create a square shaped range around each position of metros and count the number of houses within the range. The length of a side of the square is 0.04.

In:

```
print "\nFind the number of houses near each metro"
for metro in way_pos.keys():
    count = 0
    lat = way_pos[metro]["lat"]
    lon = way_pos[metro]["lon"]
    for pos in building_pos:
        if pos[0] >= lat - 0.02 and \
            pos[0] <= lat + 0.02 and \
            pos[1] >= lon - 0.02 and \
            pos[1] <= lon + 0.02:
            count += 1
    print metro, ":", count
```

Out:

```
Find the number of houses near each metro
McLean : 15
Spring Hill : 46
West Falls Church-VT/UVA : 225
East Falls Church : 381
Tysons Corner : 46
Vienna/Fairfax-GMU : 712
Dunn Loring-Merrifield : 134
Greensboro : 48
```

According to the result, there are the most number of houses near the “Vienna/Fairfax-GMU” station and the least number of houses near the “McLean” station.