OPENSTREETMAP PROJECT: DATA WRANGLING WITH MONGODB

-KISHAN CHOUDHURY

5/3/2016

LONDON, UNITED KINGDOM

The following is an analysis of the OpenStreetMap data with an objective of finding anomalies and correcting them. It also involves restructuring the data in way, which makes it more meaningful and using the MongoDB database to store the data.

REASON FOR CHOSING LONDON:

Unfortunately, the areas I am familiar with do not have enough data in OpenStreetMap. So, I had to choose a random city which has enough data suitable for analysis. So, I cannot rely on my local knowledge of the area to perform cleansing operation but I feel, such a situation is commonly faced by data analysts.

ANALYZING THE DATA TO FIND ANOMALIES:

First, we try to analyze the data to find various anomalies.

Printing the various top level tags:

CODE:

```
#defining a set to avoid duplication of values
element=set()
#parsing the file
for event,elem in ET.iterparse(SAMPLE_FILE,events=('start','end')):
    if event=='end':
        element.add(elem.tag)
        #clearing the memory
        elem.clear()
print element
```

OUTPUT:

```
set(['node', 'nd', 'bounds', 'member', 'tag', 'relation', 'way', 'osm'])
```

Analyzing Street Types:

CODE:

```
#using regular expression to find street type
street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)
#using an expected list to focus on just the unexpected ones
expected = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Place", "Square", "Lane", "Road", "Trail", "Parkway", "Commons"]
#function to find unexpected street names
#the function helps us get a list of unexpected Street types and the corresponding Street Names which we can analyze
def audit_street_type(street_types, street_name):
    m = street_type_re.search(street_name)
    if m:
        street_type = m.group()
        if street_type not in expected:
            street_types[street_type].add(street_name)
```

OUTPUT:

Note: Presenting just a few problematic outputs

```
'Ave': set(['Cherry Ave', 'Nestles Ave', 'Quintin Ave', 'Virginia Ave']),
 'Avenuen': set(['Kingston Avenuen']),
'Boadway': set(['Muswell HIll Boadway']),
'Rooad': set(['Brickfields Rooad']),
'Road)': set(['The Broadway (Ruislip Road)']),
 'Road--': set(['Weir Road--']),
'false>>': set(["<val='Arundel Avenue',<Priority; inDataSet: false, inStandard: false, selected: false>>",
                "<val='Bell Road',<Priority; inDataSet: false, inStandard: false, selected: false>>",
                "<val='Blandford Gardens',<Priority; inDataSet: false, inStandard: false, selected: false>>",
                "<val='Bradley Drive',<Priority; inDataSet: false, inStandard: false, selected: false>>",
                "<val='Woodstock Road',<Priority; inDataSet: false, inStandard: false, selected: false>>"]),
'market': set(['Exmouth market']),
 'parade': set(['Royal parade', 'purley parade']),
 'park': set(['Langdon park'])
```

Problems Identified:

Inconsistent Street Names:

Road is represented as **Rd** in most of the cases. However, some of the entries have **Road** , **Road--**,etc

Improper Cases:

Some of the entries are all in CAPS while some are all in SMALL. We need to have an uniformity.

Improper syntax:

There are some entries of the form:

"<val='Woodstock Road',<Priority; inDataSet: false, inStandard: false, selected: false>>"])

which are not in the proper syntax.

Cleansing the data:

CODE:

```
#Created a mapping dictionary to replace inconsistent street names
#including just a few to explain the process
 mapping = {"Ave" : "Avenue", "Avenuen" : "Avenue", "Saint" : "St"}
```

```
#Function to solve Capitalization problem
def capitalize(Text):
    return ' '.join(word[0].upper() + word[1:] for word in Text.split())
#Function to lookup proper street names from the mapping dictionary defined above
def update_name(name, mapping):
     name_arr=name.split(' ')
     for idx,item in enumerate(name_arr):
         if item in mapping:
             name_arr[idx]=mapping[item]
     return ' '.join(name_arr)
#A separate function to handle all the address cleaning which calls all the cleaning functions defined above
def cleanse_address(value):
    if 'false>>' in value:
        value= value.split("""")[1]
    #cleansing improper street names
    value=update_name(value,mapping)
    #capitalizing
    clean_value=capitalize(value)
    return clean_value
```

RESTRUCTURING THE DATA:

Before entering the data into MongoDB , we need to decide on the rules for restructuring. We need to wrangle the data and transform the shape of the data into a model that is suitable for storing in MongoDb. We need to transform the data according to the following rules:

- all attributes of "node" and "way" should be turned into regular key/value pairs, except
    - attributes in the CREATED array should be added under a key "created"
    - attributes for latitude and longitude should be added to a "pos" array, for use in geospacial indexing. Make sure the values inside "pos" array are floats and not strings.
- if the second level tag "k" value contains problematic characters, it should be ignored
- if the second level tag "k" value starts with "addr:", it should be added to a dictionary "address"
- if the second level tag "k" value does not start with "addr:", but contains ":", it should be split into a two-level dictionary like with "addr:".
- if there is a second ":" that separates the type/direction of a street,the tag should be ignored, for example:

CODE:

```
#Fuction for transforming the data according to our datamodel for inserting it into MongoDb
CREATED = [ "version", "changeset", "timestamp", "user", "uid"]
def shape_element(element):
    node = {}
    created={}
    pos=[]
    #processing nodes and ways
    if element.tag == "node" or element.tag == "way" :
        node['id']=element.attrib['id']
        node['type']=element.tag
        #visible attribute is not present in all the elements
        if 'visible' in element.attrib:
            node['visible']=element.attrib['visible']
        for item in CREATED:
            if item in element.attrib:
                created[item]=element.attrib[item]
        node['created']=created
        pos=[]
        if 'lat' in element.attrib:
            lat=float(element.attrib['lat'])
            pos.append(lat)
        if 'lon' in element.attrib:
            lon=float(element.attrib['lon'])
            pos.append(lon)
        if len(pos)!=0:
            node['pos']=pos
        address={}
        for item in element.iter('tag'):

            if 'k' in item.attrib:
                key_list=item.attrib['k'].split(':')

                if key_list[0]=='addr':
                    #cleaning address
                    clean_value=cleanse_address(item.attrib['v'])
                    #ignoring second and further levels of address
                    if len(key_list)>2:
                        continue
```

```
                #ignoring problematic characters
                elif problemchars.match(item.attrib['k']):
                    continue
                else:
                    address[key_list[1]]=clean_value
            else:
                #tags with : other than address
                #eliminating problematic keys
                    if '.' not in item.attrib['k']:
                node[item.attrib['k']]=item.attrib['v']
            #testing for empty dictionary
            if bool(address):
                node['address']=address
        #adding node references for ways
        if element.tag=='way':
            node_refs=[]
            for item in element.iter('nd'):
                node_refs.append(item.attrib['ref'])
            node['node_refs']=node_refs
        return node
    else:
        return None
#connecting to MongoDB
def get_db():
    from pymongo import MongoClient
    client = MongoClient('localhost:27017')
    db=client['examples']
    return db
```

OVERVIEW OF THE DATA:

After inserting the data into MongoDB, we ran some queries to find some stats on the data:

- size of the file: 2.3 GB
- number of unique users

  len(db['openstreetmap'].distinct("created.user"))

  8377 users edited the file

- number of nodes

  db['openstreetmap'].find({ "type": "node" }).count()

11011906

- number of ways

  db['openstreetmap'].find({ "type": "way" }).count()

  1565711

- number of pubs

  db['openstreetmap'].find({ "amenity": "pub" }).count()

  7361

- number of cafes

  db['openstreetmap'].find({ "amenity": "cafe" }).count()

  4607

OTHER APPLICATIONS OF THE DATASET:

The dataset may be used to find a solution to the travelling salesman problem i.e. finding the shortest path to cover all the nodes of a particular type within a particular area. However, aerial distances between the nodes cannot be considered as one cannot traverse from one node to the next in a straight line. The challenge would be in identifying the way for travelling from one node to the next and the length of the way. Once a graph constructed with vertices and edges, an algorithm can be designed to find the shortest path.