Data Wrangling with MongoDB

Razikh Shaik

October 28, 2016

1 Introduction

For the 3rd project in the Data Analyst Course from Udacity, I am tasked to assess the quality of the data and prepare it for any further analysis. The data i will be working on is provided by OpenStreetMap(Open source geographic data for the world), my area of interest was to explore the city of **Toronto**. OpenStreetMap has metro extract for Toronto readily available to download but this version of data compromises on the boundary of the city, there is a lot more data correspodning to the surroundings of the city which should not have been there. For this reason, I have custom selected and downloaded the data for the area that most likely resembles the complete city.

Here's link to the data that i will be working on Toronto data

Toronto has always been a favorite city of mine for various reasons, of which the prominent one's being the time!!! The time I have spent there with my family and friends is unparalleled. I was exploring the city's top notch one of a kind architecture, restaurants, bars, galleries, buildings and parks. City boosts a very vibrant vibe, there is something for everyone to really enjoy here. Toronto is Canada's most populous city known for its diverse culture and is considered as very strong econimical hub.

Let's get started!!

2 Data Wrangling

To get started with, lets first import all the necessary libraries. The data is in XML format, to parse the data we would require xml package.

Often times the data that we are dealing with may be quite huge which makes it not possible to load the entire data in to the memory. Making use of iterative parsing can help overcome this issue.

```
In [1]: # importing all the required libraries
        import xml.etree.cElementTree as ET
        from collections import defaultdict
        import pprint
        import re
        import codecs
        import json
        from bs4 import BeautifulSoup
        import urllib2
```

```
from pymongo import MongoClient
import os


# the xml file that i downloaded from OpenStreetMap
source_xml_file = "toronto_canada.osm"
```

2.1 Audit Tags

XML contains various tag elements to distinguish one from another. Let's figure out which are the most used tags in the data.

```
In [3]: tag_dict =  count_tags(source_xml_file)
        print_dict_by_values(tag_dict)
```

```
tag - 1580816
nd - 1487924
node - 1291949
way - 223417
member - 69451
relation - 2895
bounds - 1
note - 1
meta - 1
osm - 1
```

There is no unusual stuff going on with tags, all look to be perfectly alright as per the wiki page.

2.2 Audit Keys Type

All the elements in the XML file have keys, these keys are associated with values which provide the information on that particular node.

Let's audit for any problematic keys and what kind of various keys we have.

```
In [5]: keys_type, key_count = process_map(source_xml_file)

        pprint.pprint(keys_type)
```

```
{'lower': 938554,
 'lower with colon': 617526,
 'other': 24736,
 'problemchars': 0}
```

From the above output, there are no problematic keys that we have to deal with.

Let's look at the unique keys and how many times they are repeated.

```
In [6]: print_dict_by_values(key_count)
```

```
source - 285114
addr:street - 159311
addr:housenumber - 158833
addr:city - 140086
highway - 112109
addr:interpolation - 68713
name - 65714
surface - 53120
lanes - 44810
building - 44442
```

OSM allows the map to include an unlimited number of attributes describing each feature. The community agrees on certain key and value combinations for the most commonly used tags, which act as informal standards.

Let's Audit how many of the keys present in the data are of informal standards as mentioned on the wiki website

```
In [22]: html_page = "https://wiki.openstreetmap.org/wiki/Map_Features"

         # Scraping the data from internet to validate how many keys that
         # are in the data are primary features!
         def extract_data(page):
             ...
             return primary_keys

         primary_keys_wiki = extract_data(html_page)
         list(primary_keys_wiki)[0:5]

Out[22]: ['shop', 'seasonal', 'maxspeed', 'office', 'healthcare:speciality']
```

The output displayed above are the primarily used key types.

```
In [13]: print "Keys that are not listed as Primary: " + str(len(not_in_primary_key
         print "Keys that are listed as Primary: " + str(len(in_primiary_keys))

Keys that are not listed as Primary: 620
Keys that are listed as Primary: 410
```

It becomes a daunting task to validate the 620 keys as the OSM provides flexibility to user to create custom keys. There were many spelling mistakes and inappropriate data that is entered. If there is a formal list of standard keys, these keys can be validated against them to keep accurate data.

2.3 Audit Users

In this step, I want to audit uid to see if they are actually all machine genereted intergers with no formatting errors.

```
In [14]: # This function returns the user id from xml element
         def get_user(element):
             return element.get("uid")

         # This function checks for all the user id and if there
         # are any formatting issue with uid and any incorrect uid
         def process_users(filename):
             ...
             return [users, invalid_uid]

In [15]: users, invalid_users=process_users(source_xml_file)
         print "Out of " + str(len(users)) + " , There are "+
               str(len(invalid_users)) + " invalid uid's"

Out of 1250 , There are 0 invalid uid's
```

3 Problems Encountered in the Map

3.1 Audit/ Clean Street Type

The first problem encountered in the map is the one about the street type. As we can see in the following code, words can become abbreviations or just a uppercase word instead of a lowercase word. There were spelling mistakes in the data as well.

```
In [28]: #st_types = audit_streets(source_xml_file)
         for key in st_types.keys()[0:3]:
             print key + " :  "
             pprint.pprint(st_types[key])

Ridge :
set(['Basking Ridge',
     'Bayview Ridge',
     'Circle Ridge',
     'Dianawood Ridge',
     'Echo Valley Ridge',
     'Hunting Ridge',
     'Orchard Haven Ridge'])
Hills :
set(['Barkdene Hills', 'Cobble Hills'])
Cottages :
set(['Wellesley Cottages'])


In [31]: # Loops over all the streets and checks if there is a better
         # name for the street(get rid of typos and abbreviations)
         for st_type, ways in st_types.iteritems():
             if st_type in mapping.keys():
                 for name in ways:
```

```
            better_name = update_name(name, mapping)
            print name, "=>", better_name
            name = better_name
```

Spadina Rd => Spadina Road
Kennedy Rd => Kennedy Road
JARVIS STREET => JARVIS Street
Ryerson avenue => Ryerson Avenue
Dovercourt => Dovercourt Road
Red Robinway => Red Robin Way
Sea Robinway => Sea Robin Way
Dundas street => Dundas Street
San Robertoway => San Roberto Way

3.2 Audit / Clean PostCode

There were formatting typos for postal code. Most of them were correct, only a one refered to postal code from the city of Ottawa instead of Toronto.

A few mistakes were rectified by manual searching for the address.

```
In [20]: audit_postcode(source_xml_file)
```

```
M5T 1R9, M1P 2L7
M36 0H7
L4K
M5J 2G
K4A 1W9
```

M5T 1R9, M1P 2L7 — M1P 2L7 is actually the wrong address
M36 0H7 - should be replaced with M3C 0H7 invalid postal code
L4K - vaughan area code excluded
M5J 2G to be replaced with M5J 2G8- invalid, This tag is associated to a way which is wrong!
K4A 1W9 - ottawa area excluded, This tag is associated to a way which is wrong!

3.3 Audit Bank

From the analysis done, I see that all the Bank information has been specified correctly. The only concern here is there should be an additional tag to hold the information of the parent bank company. So, when there is TD Bank in 10 different locations with different names, we can know by the parent tag that they are all related to TD Bank.

```
In [39]: audit_banks(source_xml_file)
         #unique_banks
```

defaultdict(set,
{'BMO': {'BMO', 'BMO - Bank of Mointreal','BMO - Bank of Montreal', 'BMO Bank of Montreal', 'BMO Bank of Montreal/BMO Nesbitt Burns', 'BMO Financial Group', 'BMO Insurance', 'BMO Nesbitt Burns'},

'BPI': {'BPI'},
...
'TD': {'TD', 'TD Bank Drive Through', 'TD Canada Trust', 'TD Commercial Banking', 'TD Waterhouse'},
...
'iTRADE': {'Scotia iTRADE'}})

In the output above I have written the code to group the banks, It would have been much easier if there was a tag that will hold the parent company name.

3.4 XML to JSON Conversion

It is now time to convert the XML data to JSON and thereby loading it into mongodb to do further assessment.

I have used more structured json format as below to keep data better organized.

{
"id": ,
"type": ,
"pos": ,
"created": { "uid": , "changeset":, "version": , "user": , "timestamp": }
"features" : { # all tag related information }
"node_refs" : []
}

3.5 Connect to MongoDB

The data that is obtained from the previous step can be loaded into mongo via pymongo. This can also be acheived by using mongoimport to import the data from JSON file to mongodb.

```
In [40]: # Default connection to localhost
         client = MongoClient()
         # switch to project DB
         db=client.project

         # drop any data if already present in toronto collection
         db.toronto.drop()
         # insert all data
         db.toronto.insert_many(data)
```

3.5.1 Data Overview

File size (in bytes)

```
In [41]: os.path.getsize(source_xml_file)
```

```
Out[41]: 336400599L
```

```
In [42]: os.path.getsize(source_xml_file+".json")
```

```
Out[42]: 364747302L
```

OSM file size: 320 MB
JSON file size: 356 MB

Number of documents

```
In [43]: nb_doc=db.toronto.find().count()
         print(nb_doc)
```

```
1511572
```

Number of nodes

```
In [44]: nb_nodes=db.toronto.find({"type":"node"}).count()
         print(nb_nodes)
```

```
1290711
```

Number of ways

```
In [45]: nb_way=db.toronto.find({"type":"way"}).count()
         print(nb_way)
```

```
220861
```

Number of unique users

```
In [46]: nb_unique_users=len(db.toronto.distinct("created.user"))
         print(nb_unique_users)
```

```
1154
```

Top 1 contributing User

```
In [48]: cursor=db.toronto.aggregate(
             [
                 {"$group":{"_id":"$created.user", "count":{"$sum":1}}},
                 {"$sort":{"count":-1}}, {"$limit":1}
             ]
         )
         for res in cursor:
             user1=res["_id"]
             user1_count=res["count"]

         print "Top contributor is "+ user1 + " with - " + str(user1_count) + " cor
```

```
Top contributor is andrewpmk with - 1187521 contributions.
```

Number of users appearing only once (having 1 post)

```
In [50]: user_1post=db.toronto.aggregate(
             [
                 {"$group":{"_id":"$created.user", "count":{"$sum":1}}},
                 {"$sort":{"count":-1}},
                 {"$match":{"count":1}},
                 {"$group":{"_id":"null","total":{"$sum":"$count"}}}
             ]
         )

         for res in user_1post:
             nb_user_1post=res["total"]

         print "There are " + str(nb_user_1post) +  " users who only contributed or
```

```
There are 288 users who only contributed once.
```

Number of unique sources

```
In [52]: len(db.toronto.distinct('feature.source'))
```

```
Out[52]: 233
```

Top 5 sources

```
In [54]: user_1post=db.toronto.aggregate(
             [
                 {"$group":{"_id":"$feature.source", "count":{"$sum":1}}},
                 {"$sort":{"count":-1}},
                 {"$limit":5}
             ]
         )

         for res in user_1post:
             pprint.pprint(res)
```

```
{u'_id': None, u'count': 1227435}
{u'_id': u'CanVec 6.0 - NRCan', u'count': 174462}
{u'_id': u'Bing', u'count': 35257}
{u'_id': u'StatCan 92-500-X', u'count': 27284}
{u'_id': u'Geobase_Import_2009', u'count': 15395}
```

3.5.2 Additional data exploration using MongoDB queries

Top 10 amenities

```
In [56]: top_amenity=db.toronto.aggregate(
             [
                 {"$match":{"feature.amenity":{"$exists":1}}},
                 {"$group":{"_id":"$feature.amenity","count":{"$sum":1}}},
                 {"$sort":{"count":-1}},
                 {"$limit":10}
             ]
         )

         for res in top_amenity:
             pprint.pprint(res)
```

```
{u'_id': u'parking', u'count': 11592}
{u'_id': u'fast_food', u'count': 1520}
{u'_id': u'bench', u'count': 1473}
{u'_id': u'restaurant', u'count': 1408}
{u'_id': u'post_box', u'count': 1101}
{u'_id': u'cafe', u'count': 838}
{u'_id': u'waste_basket', u'count': 777}
{u'_id': u'place_of_worship', u'count': 731}
{u'_id': u'bank', u'count': 597}
{u'_id': u'school', u'count': 467}
```

Top 5 fast-food chain

```
In [64]: top_fastfood=db.toronto.aggregate(
             [
                 {"$match":{"feature.amenity":"fast_food"}},
                 {"$group":{"_id":"$feature.name","count":{"$sum":1}}},
                 {"$sort":{"count":-1}},
                 {"$limit":5}
             ]
         )

         for res in top_fastfood:
             pprint.pprint(res)
```

```
{u'_id': u'Subway', u'count': 200}
{u'_id': u'Pizza Pizza', u'count': 99}
{u'_id': u"McDonald's", u'count': 68}
{u'_id': u'Mr. Sub', u'count': 36}
{u'_id': u'Pizza Nova', u'count': 34}
```

List of top 5 cuisine

```
In [65]: cuisine = db.toronto.aggregate(
             [
```

```
                {"$match":{"feature.cuisine":{"$exists":1}}},
                {"$group":{"_id":"$feature.cuisine", "count":{"$sum":1}}},
                {"$sort":{"count":-1}},
                {"$limit":5}
            ]
        )
        for res in cuisine:
            pprint.pprint(res)
```

```
{u'_id': u'coffee_shop', u'count': 506}
{u'_id': u'pizza', u'count': 267}
{u'_id': u'sandwich', u'count': 235}
{u'_id': u'burger', u'count': 176}
{u'_id': u'chinese', u'count': 82}
```

Cuisine tag has not relevant information in it. The only relevant cuisine from the above results is chinese.

Biggest Bank

```
In [67]: temp=db.toronto.aggregate(
            [
                {"$match":{"feature.amenity":"bank"}},
                {"$group":{"_id":"$feature.name","count":{"$sum":1}}},
                {"$sort":{"count":-1}},
                {"$limit":3}
            ]
        )

        for res in temp:
            pprint.pprint(res)
```

```
{u'_id': u'TD Canada Trust', u'count': 133}
{u'_id': u'Scotiabank', u'count': 92}
{u'_id': u'CIBC', u'count': 64}
```

Royal Street Names

```
In [110]: royal_street_names = db.toronto.distinct(
             "feature.address.street", {
                 "feature.address.street": {
                     "$regex" : "King|Queen|Prince|Princess|Royal"
                 }
             }
         )

         print "There are " + str(len(royal_street_names)) + " of them."
```

```
        print "Here are a few - "
        for i, street in zip(range(1,6), royal_street_names[0:5]):
            print str(i) + ") " + street
```

```
There are 140 of them.
Here are a few -
1) Queen Street West
2) King Street West
3) Queen's Quay West
4) King Street East
5) Prince Arthur Avenue
```

3.5.3 Additional Ideas

Contributor statistics Let's find out how much percentage of contributions made by top 5 contributors

```
In [104]: top_5_users = db.toronto.aggregate(
              [
                  {"$group":{"_id":"$created.user", "count":{"$sum":1}}},
                  {"$sort":{"count":-1}}, {"$limit":5}
              ]
          )

          all_users_contribution = db.toronto.aggregate(
              [
                  {"$group":{"_id":"$created.user", "count":{"$sum":1}}},
                  {"$sort":{"count":-1}}
              ]
          )
          total_contributions_made = 0
          for user_con in all_users_contribution:
              total_contributions_made += user_con["count"]
```

```
In [105]: contributions_made_by_top_5 = 0
          for res in top_5_users:
              contributions_made_by_top_5 += res["count"]
              print "User " + res["_id"] + " has " + str(round(float(res["count"]) *

          print "\nTop 5 users contribution is  " + str(round(float(contributions_m
```

```
User andrewpmk has 79.0% contribution
User Kevo has 5.0% contribution
User Mojgan Jadidi has 2.0% contribution
User andrewpmk_imports has 2.0% contribution
User Bootprint has 1.0% contribution

Top 5 users contribution is  89.0%
```

4 Conclusion

The map about the city of Toronto is relatively clean so I could retrieve some interesting content. But still the data is not entirely clean. The data contains some mistakes or different references for the same feature (like abbreviation or slightly different names). So I had to clean the data programmatically for the street and the postal code. But they are not the only features, that require cleaning. There are many tags for the key types that are ambiguous, the values entered are not appropriate as well. For example when looking at the cuisine, most the data there was entered incorrectly.

Ideas to improve data quality of OSM:

When we audit the data, it was very clear that although there are minor error caused by human input, the data is fairly well-cleaned. Considering there're hundreds of contributors for this map, there is a great numbers of human errors in this project. OSM gives a lot of flexibility for the user to enter new information. The data that is entered should agree with strong guidelines. I'd recommend a srtuctured input form so everyone can input the same data format. Evolving this form to allow additional information should be keenly monitored.

Potential Cost of Implementation:

There're few potential issues could you see that may arise from the implementation of this solution. One of which is the amount of effort to engineer all these processes and the cost of creating, auditing & maintaining these initiatives could be so overwhelm and require a dedicated team responsible for all these projects.