Udacity - Data Wrangling with MongoDB

Project 3 - OpenStreetMap Project

Amodiovalerio Verde (amodiovalerio.verde at gmail.com)

Problems encountered in map

For this project, we used an XML OpenStreetMap data file relative to Milan (Italy). Specifically, we had cleaned and audited the data file, reshaped the data, exported as a JSON, imported the data into MongoDB and queried the database to gather some information

The data file was indeed quite clean, with very few problems related to data itself like:

- italian street names format, where street denomination is the first part of an address
- wrong case in street names
- typo in some 'addr:' values
- attributes 'type' that overwrite node 'type'
- problem characters in some values

We faced also some different problems, not related to data itself, that we had to handle during the project were related to:

- size of file. we had to execute clear() each time we finish to process a node or we could run out of memory
- non unique ids in nodes
- excessive specialization of tags

1. Acquiring data files (getdata.py)

For this project, as we said, we choose to use the map of the city of Milan in Italy and the reason for this choice is that because since Milan is the city where I live, it was easier to manage data and spot any incongruence or error. Moreover, knowing the local notation for elements like street names, postcodes, prefix numbers and so on, helped us to decide how to handle wrong data

We choose to get also a smaller map to speed up coding and testing and then apply everything to the larger map

getdata.py takes cares of downloading the two files (that are not included in project submission)

```
Downloading http://overpass-api.de/api/map?bbox=9.0603,45.3808,9.2728,45.5256
Download complete. File size: 185553321

Downloading http://overpass-api.de/api/map?bbox=9.1635,45.4507,9.2076,45.4824
Download complete. File size: 25839325
```

Please note that the small data file was used only for testing purpose. All information in this document is related specifically to the bigger data file

The size of our data file, that we are going to review, transform and import, is about 185 Mbyte

2. First analysis of data files (mapparser.py)

First, we used **mapparser.py** to have an overview of the data, see how many elements there are in the file, and check if the XML is valid and if we can parse it successfully

```
Number of levels: 3
```

```
Level: 0
   osm [1]
Level: 1
   node [752505], bounds [1], note [1], meta [1], relation [4361], way [133735]
Level: 2
   member [92045], tag [560838], nd [962776]

Total number of nodes: 2506264
```

As we can see, the file is a valid XML, it contains 2,506,264 nodes and its structure has three levels of depth

Most of first level nodes are of type 'node' or 'way', and these are the nodes we are interested in

Both type of nodes have this set of common attributes: ('changeset', 'uid', 'timestamp', 'version', 'user', 'id') while 'node' has two additional attributes: ('lat', 'lon')

```
Node: node
{'changeset': 752505, 'uid': 752505, 'timestamp': 752505, 'lon': 752505, 'version':
752505, 'user': 752505, 'lat': 752505, 'id': 752505}

Node: way
{'changeset': 133735, 'version': 133735, 'uid': 133735, 'timestamp': 133735, 'id':
133735, 'user': 133735}
```

There are no node with missing attributes

We can notice an 'id' attribute for each node (both 'node' and 'way'); later we will see if we can use it as an id for MongoDB

3. Tag analysis (tags.py)

We were interested in nodes 'tag' and especially in their keys/values, so we ran **tags.py** that shows us count and the list of all values for tag keys ('k') found by type

```
{'lower': 481120, 'lower_colon': 77412, 'other': 2302, 'problemchars': 4}
```

The problemchars are:

```
{'comment.it:2': 1, 'step.condition': 3}
```

In this case the problem is the char '.' as it is used by MongoDB to query subdocuments

We can also see (from the list of lower colon tag keys) that 'comment.it:2' should be 'comment:it:2'

So we substitute '.' with ':' and for step.condition (that is a legit proposed OSM feature), we replace '.' with '_'

We can also notice an excessive tag specialization with a relatively high number of tags has just few occurrences. Some of these tags seems also not in the official OSM set

We can also observe that there are other field with colon that could be grouped in dictionaries as we will do with 'addr:xxx' keys i.e. 'building', 'contact' or 'payments'

```
{'building:color':3, 'building:colour':2, 'building:levels':3011,
'building:material':7, 'building:min_level': 8, 'building:part': 232, 'building:use':
35}
```

A deeper analysis is needed because most of them (differently for 'addr:') have also a value in the main tag, so we handled this values replacing ':' with '__' (double underscore)

Browsing the list of lower case tags, we also notice that there is a tag named 'type'. This will be a problem, because we need this key name to store the information about the node type; we will rename the tag 'type' with 'k_type' to keep the information

4. Explore data (users.py)

With **users.py** we get some information on our data; this also came handful later, to check that the whole file has been processed and imported in MongoDB

```
Unique 'k' values: 856
Unique users: 1086
Unique buildings: 66
Unique addr:city: 21
Unique addr:street: 2103
Unique addr:postcode: 54
Unique amenities: 107
```

Looking inside the exported **osm_unique_tags.json** we can clearly see that probably there are other things to check and fix in various tags:

- some keys are split due a typo:

```
(..., 'unknown', 'unkown', ...)
```

- some keys are split because of both singular and plural names:

```
(..., 'garage', 'garages', ...)
(..., 'watch', 'watches', ...)
(..., 'clock', 'clocks', ...)
```

- some keys have the same meaning:

```
'chemist': set(['cosmetic',
                'cosmetics',
                'hairdressing',
                'perfume',
                'perfume;cosmetics',
                'purfume'])
```

- some keys has wrong format (i.e. italian prefix is +39 in phone numbers):

```
(..., '+30 02 33001882', '+30 02 63470553', '+30 02 76394975', ...)
```

- some keys represent arrays but with different formats:

```
(..., 'pizza,_italiana', 'pizza,italian', 'pizza:italian', 'pizza; italian', ...)
```

A deeper analysis should be conducted on each keys to do some serious fix/cleaning. For the scope of this project, we will focus on just some elements to be audited and fixed

5. Audit street names and other data (audit.py)

We then use **audit.py** to dump all addr:keys counts and calculate how many street names we are going to rename

Data file is quite clean and we have to fix just 63 street names, most of them just because of wrong case or a typo

```
Renamed due mapping: 48
Renamed due expected: 15
```

We also decide to drop some values like addr:state, addr:floor, addr:full, addr:unit because of the low number of nodes affected (1 or 2) or because they are misused (like addr:state to indicate a region)

```
u'floor': {u'1': 1, u'3': 1, u'4': 1},
u'full': {u'Corso Buenos Aires angolo Viale Tunisia': 1},
u'state': {u'Lombardia': 1, u'MI': 1},
u'unit': {u'A': 1, u'P01': 1},
```

Finally, we find some minor typo in other keys, and we are going to fix them too

- 'addr:postcode' contain two non-valid postcodes --> 20100 (remove) and 2090 (rename in 20090)
- 'addr:city' contain 'milano' --> rename in 'Milano'
- 'addr:province' contain an abbreviation --> rename 'MI' in 'Milano'

6. Additional checks (uniqueid.py)

We notice that we have a 'id' that seems unique and that we can use as our unique key in MongoDB

We run **uniqueid.py** to check if 'id' is unique but unfortunately we discover that there are duplicates

```
'id' found: 886240
Unique 'id': 886216
Duplicate 'id': 12
```

The ids in an OSM file are not unique, and this is also confirmed on http://wiki.openstreetmap.org/wiki/Node, where it says:

'Node ids are unique between nodes. (However, a way or a relation can have the same id number as a node.)'

7. Data cleaning and shaping (data.py)

We run **data.py** to shape and filter the OSM XML file and create the JSON file we will import into MongoDB

The following fix/requirement were applied on OSM XML file before dumping the JSON:

[Req.1] We will process only 2 types of top level nodes: 'node' and 'way'; other top level nodes will be discarded
[Req.2] Set 'type' with value = node type ('node' or 'way')
[Req.3] Attributes for latitude ('lat') and longitude ('lon') should be added to a 'pos' array as float
[Req.4] Attributes in the CREATED array should be grouped under a key 'created'
[Req.5] All other attributes of 'node' and 'way' should be turned into regular key/value pairs
[Req.6] If 'k' in 'tag' node contain problematic char, it will be dropped
[Req.7] If 'k' in 'tag' node in TAGDROP it will be dropped
[Req.8] If k attributes is in the form addr:x it will converted in an array 'address'
[Req.9] If k attributes is named 'type', rename it with 'k_type'
[Req.10] If k attribute is named 'comment.it:2', rename it with 'comment:it:2'
[Req.11] If k attribute is named 'step.condition', rename it with 'step_condition'
[Req.12] If node contains nd tag, there will be transformed into an array
[Req.13] Postcodes will be fixed
[Req.14] Fix 'addr:city' for milano --> Milano
[Req.15] Fix 'addr:province' for MI --> Milano
[Req.16] If the second level tag 'k' value does not start with 'addr:', but contains ':', replace ':' with '__'
[Req.17] Fix 'addr.street' using STREETMAPPING

*Note that you can find the [Req.xx] reference to the specific code in **data.py***

Resulting JSON is about 200 Mbyte, little bigger than the original XML file (185 Mbyte)

8. Importing and check data (checkdata.py)

We import our JSON file via mongoimport:

```
mongoimport --db udacity --collection milan --drop --file milan_italy_big.osm.json
```

The import successfully imported 886.240 documents

Then we check if everything was imported, using **checkdata.py**

```
Nodes in db (with type 'node'): 752505 Correct
Nodes in db (with type 'way'): 133735 Correct
Total nodes in db: 886240 Correct
Unique users: 1086 Correct
Unique buildings: 66 Correct
Unique addr:city 20 Correct, we rename one key to an already existent one
Unique addr:street 2103 Correct
Unique postcode: 52 Correct, we delete one key and rename another one
Unique amenity 107 Correct
```

Overview of the Data

Dataset size

```
> db.milan.find().count()

886240
```

Number of contributors

```
> db.milan.distinct('created.user').length

1086
```

Numbers of documents with type='node'

```
> db.milan.find({'type':'node'}).count()

752505
```

Top 5 user by contributors

```
> db.milan.aggregate([{'$group':{'_id':'$created.user','count':{'$sum':1}}},{'$sort':
{'count':-1}},{'$limit':5}])

{u'count': 180111, u'_id': u'ilrobi'}
{u'count': 148736, u'_id': u'adirricor'}
{u'count': 79852, u'_id': u'Alecs01'}
{u'count': 78112, u'_id': u'fedc'}
{u'count': 39930, u'_id': u'Cristian1989'}
```

Number of user with only 1 contribution

```
> db.milan.aggregate([{'$group':{'_id':'$created.user','count':{'$sum':1}}},
{'$sort':{'count':-1}}, {'$match':{'count':{$eq:1}}}]).itcount()

314
```

About 29% of users just made a single contribution. This may suggest that the submission process is not easy or user-friendly

Number of unique amenities

```
> db.milan.distinct('amenity').length

107
```

Top 10 amenities

```
> db.milan.aggregate([{'$match':{'amenity':{'$exists':1}}},{'$group':{'_id':'$amenity',
'count':{'$sum':1}}}, {'$sort':{'count':-1}}, {'$limit':10}])

{u'count': 2235, u'_id': u'parking'}
{u'count': 1390, u'_id': u'bench'}
{u'count': 1250, u'_id': u'waste_basket'}
```

```
{u'count': 1101, u'_id': u'restaurant'}
{u'count': 795, u'_id': u'cafe'}
{u'count': 550, u'_id': u'bicycle_parking'}
{u'count': 526, u'_id': u'bank'}
{u'count': 439, u'_id': u'drinking_water'}
{u'count': 420, u'_id': u'bar'}
{u'count': 406, u'_id': u'school'}
```

A lot of parking means that there are a lot of cars in Milan.
But we can also notice how Milan is a financial city

Number of unique buildings
```
> db.milan.distinct('building').length

66
```

Top 10 buildings
```
>db.milan.aggregate([{'$match':{'building':{'$exists':1}}},{'$group':
{'_id':'$building','count':{'$sum':1}}}, {'$sort':{'count':-1}}, {'$limit':10}])

{u'count': 28091, u'_id': u'yes'}
{u'count': 6173, u'_id': u'apartments'}
{u'count': 2152, u'_id': u'house'}
{u'count': 1493, u'_id': u'residential'}
{u'count': 1266, u'_id': u'industrial'}
{u'count': 552, u'_id': u'roof'}
{u'count': 482, u'_id': u'commercial'}
{u'count': 309, u'_id': u'school'}
{u'count': 302, u'_id': u'garages'}
{u'count': 291, u'_id': u'office'}
```

Number of unique shops
```
> db.milan.distinct('shop').length

159
```

Top 10 shops
```
> db.milan.aggregate([{'$match':{'shop':{'$exists':1}}},{'$group':{'_id':'$shop',
'count':{'$sum':1}}}, {'$sort':{'count':-1}}, {'$limit':10}])
```

```
{u'count': 555, u'_id': u'clothes'}
{u'count': 309, u'_id': u'supermarket'}
{u'count': 174, u'_id': u'hairdresser'}
{u'count': 171, u'_id': u'kiosk'}
{u'count': 157, u'_id': u'bakery'}
{u'count': 121, u'_id': u'shoes'}
{u'count': 111, u'_id': u'car_repair'}
{u'count': 90, u'_id': u'florist'}
{u'count': 86, u'_id': u'newsagent'}
{u'count': 78, u'_id': u'jewelry'}
```

All these clothes, hairdresser, shoes, jewelry can just confirm that Milan is a fashion capital

Number of unique cuisines
```
> db.milan.distinct('cuisine').length

82
```

Top 10 cuisines
```
> db.milan.aggregate([{'$match':{'amenity':'restaurant'}},{'$match':{'cuisine':
{'$exists':1}}},{'$group':{'_id':'$cuisine','count':{'$sum':1}}}, {'$sort':
{'count':-1}}, {'$limit':10}])

{u'count': 160, u'_id': u'italian'}
{u'count': 148, u'_id': u'pizza'}
{u'count': 57, u'_id': u'regional'}
```

```
{u'count': 32, u'_id': u'japanese'}
{u'count': 32, u'_id': u'chinese'}
{u'count': 26, u'_id': u'sushi'}
{u'count': 13, u'_id': u'indian'}
{u'count': 12, u'_id': u'seafood'}
{u'count': 11, u'_id': u'italian;pizza'}
{u'count': 8, u'_id': u'burger'}
```

Looking at top 20, that includes also 'asian', 'mexican', 'greek', 'korean' and 'spanish', we can state that in Milan one can really taste cuisine from all over the world

Number of unique postcodes

```
> db.milan.distinct('address.postcode').length

52
```

Top 10 postcodes

```
> db.milan.aggregate([{'$match':{'address.postcode':{'$exists':1}}},{'$group':{'_id':
'$address.postcode','count':{'$sum':1}}}, {'$sort':{'count':-1}}, {'$limit':10}])

{u'count': 1919, u'_id': u'20090'}
{u'count': 1138, u'_id': u'20121'}
{u'count': 1104, u'_id': u'20124'}
{u'count': 625, u'_id': u'20125'}
{u'count': 520, u'_id': u'20154'}
{u'count': 507, u'_id': u'20133'}
{u'count': 456, u'_id': u'20159'}
{u'count': 447, u'_id': u'20131'}
{u'count': 422, u'_id': u'20122'}
{u'count': 328, u'_id': u'20123'}
```

Other ideas about the datasets

We identified some problem with the dataset:

- Typos and wrong case
- Too much specialized tags
- No strict control on data submission

Typo/Wrong case

It would be nice first do an analysis overall dataset (world.osm) to see how tags are used. With so much data, it will be easier to identify errors and typos. There are already some similar projects like http://taginfo.openstreetmap.org/keys, but it would be nice to have - for example - the possibility to filter by country or other elements

In order to fix typos and wrong cases, dictionaries of known entities could be used to match, validate and, in certain extent, correct the data in input; dictionaries can be built using the OSM data, values that occur often, most of times (but not always) are the correct ones

Benefits:
- Can programmatically correct typo (or suggest the right form)
- Prevent keys duplication due typos

Anticipated problem:
- Global dictionary creation
- The fact that there are few occurrences of a value, doesn't mean automatically that the value is wrong

Official tag constrain

To avoid specialized tags, it should be possible to not permit the use of key tags not OSM compliant/suggested but this is 'against' the scope of OSM that states: OpenStreetMap's free tagging system 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

We can still build a dictionary of 'official' keys and warn user about the presence of an 'unofficial' keys; or keys that are not 'official' can be tagged to skip them during queries

Benefits:

- Prevent tag over-specialization
- Can help choose the right tag

Anticipated problems:

- OSM will never force user to use only 'official' tags
- Official tags change with time
- There should be always available an updated dictionary of 'official' tags

Coverage indicator

Another thing that could improve the contribution, but also the adoption of OSM in apps and other applications, is a coverage indicator

A coverage indicator will tell how much (more or less) a city is covered by OSM data. Using aggregate data on number of ways, number of restaurants, number of schools,... it should be possible to have a rough indication of how much job need to be done to have a good coverage.

Another aspect to consider, is the number of submissions by postcode. This way it would be possible to see - graphically on a map - which local area need more contributions.

Benefits:

- a willing contributor can see which part of a map needs more data
- any users of OSM data, can see how much of the area they're looking for is covered by OSM data

Anticipated problems:

- it should be not easy to identify a good proxy for coverage. It is possible to have a rough estimate looking at the total number of buildings/streets/shop of a city/area and see how much of the elements in total number, are in the map
- postcode (and their coverage) are not standard. They can cover smaller or larger area, depending on country, city (at least in Italy), …

Webliography

- http://wiki.openstreetmap.org/wiki/Key:addr
- https://docs.python.org/2/library/re.html
- http://gis.stackexchange.com/questions/103572/are-osm-ids-unique-over-all-object-types
- https://docs.python.org/2/library/xml.etree.elementtree.html
- https://alexlouden.com/posts/2015-defaultdict-in-python.html
- https://docs.python.org/3/library/collections.html
- http://stackoverflow.com/questions/15644964/python-progress-bar-and-downloads
- http://stackoverflow.com/questions/12925052/python-and-default-dict-how-to-pprint
- http://stackoverflow.com/questions/613183/sort-a-python-dictionary-by-value
- http://stackoverflow.com/questions/6048085/writing-unicode-text-to-a-text-file
- http://stackoverflow.com/questions/2104080/how-to-check-file-size-in-python and other pages on stackoverflow