Submission document : Wrangle OpenStreetMap Data

The Data

First step , given the experience we got with the case study, is to familiarize with the data.

Data , in this case, is in XML format. The riginal file downloaded from OSM metro extracts for the city of Buenos Aires, were I live, is around 400 MB in size so I reduced the file to a sample of 5 mb created using dd (count=20000) .

The Buenos Aires subway is the 1st subway system in Latinamerica and I used it a lot so my first guess was how the OSM project managed to describe subway stations and lines in general in maps.

My initial idea was to extract subway information from the provided OSM file , insert it in SQL and create some reports and statistics based on that. What I actually found was , as we saw in the course, that the data was not complete or with the expected details.

Subway information in the provided OSM extract is aggregated in Node elements. Here an example:

```
<node id="256241854" lat="-34.639835" lon="-58.458014" version="13" timestamp="2017-05-03T14:31:10Z" changeset="48367783" uid="51045" user="Geogast">
    <tag k="name" v="Varela"/>
    <tag k="public_transport" v="stop_position"/>
    <tag k="railway" v="station"/>
    <tag k="start_date" v="1985-10-31"/>
    <tag k="station" v="subway"/>
    <tag k="subway" v="yes"/>
    <tag k="wikidata" v="Q3296869"/>
</node>
```

So initially I found two things to fix :

a) The “name” of the stations was not normalized. The official reference of the underground system is http://www.buenosaires.gob.ar/subte/mapa-y-combinaciones but they only provide a JPG file for users to review (mapa_subte.jpg). The name of the stations on that official picture are different from the one I found in the OSM file (at least some of them).

b) No information relates the stations with their corresponding lines. There are 7 subway lines in Buenos Aires identified by the following letters (A,B,C,D,E,H and P).

The Data auditing process

So given findings a and b above I basically had to go through the following actions :

1) Provide a programmable reference to correct data .

As I mentioned in point “a” the city of Buenos Aires only provide a JPG image of the whole map of lines and subway stations. Since some of the stations names are not the official ones in the OSM map I needed a normalized reference. So, basically I packaged a data structure of official station names and lines (based on JPG info) and a couple of methods to obtain the whole list of correct names or the list of stations by line (see : normalized_stations.py)

2) Sample the OSM file and clean station names

As I already mentioned original extracted file from OSM project is about 400 Mb (buenos-aires_argentina.osm) . The total number of stations in the underground system based on the official information is 104 . I sampled the original file to create a reduced size file of about 5 Mb (bs_as_subway.osm). This reduced files has 63 stations. My proposed work around this 63 stations is :

2.a) Parse the reduced osm file and process each node (parse_and_collect_nodes routine) for start and end events for node “tags”.

2.b) Call process_every_node routine to extract ‘k’/’v’ pairs for every node isolated by the first routine.

2.c) Call extract_stations to filter nodes where tags contain the “subway” key. Not all the Nodes in the OSM file mark subway stations. So, instead of moving forward processing all the nodes the idea here was, before extracting the names, to filter those Nodes without subway info. If the Node actually contains subway info, I append its “name” tag to the “actual_stations” list.

2.d) Finally , since I used ascii coding to create my structure of the official stations name, I use a routine to decode actual_stations from unicode to ascii on to_ascii_actual_stations so they can be comparable. They build up a dictionary with an index and the station name after the unidecode process.

3) First pass stations and the rest

A first pass of the actual_stations collected from OSM through a compare with official stations (split_match_nomatch_stations routine) provide a mixed results : some of the stations are exact match and some of them dont . The first group is registered in the match_stations dictionary. The second group , in the nomatch_stations dictionary.

4) Dealing with the “no match” stations names

To deal with the stations names that dont match official names I had to “tokenize” the OSM names and copare the tokens , any of them, to official names. The reasson is that I could not found other way to match -neither a way to do it with regular expressions- because the names is composed of more than

one word and some of them could match and others dont. So, first of all I dont want insignificant tokens like “de”, “los”, “la” (which are spanish terms for “of”, “the”, etc) neither line letters (like “C” or “D”) attached to the name so I use a split_nomatch outine to split the string with the name in tokens and ignore those tokens_to_ignore.

To continue with this process the idea is to use the power of sqlite3.

So , I first load several dictionaries to move them later to csv and from there to sqlite3. With the information of the different group of stations I created the following dictionaries, csv files and sqlite tables :

Dictionary	Description	CSV file	Sqlite table
final_review_nomatch	Dictionary with a decoded ID , the station name that did not match, the probable match for tokens and a counter	nomatch_tokens.csv	nomatch_tokens
nomatch_stations_dict_list	Dictionary with stations that did not match	nomatch.csv	nomatch_stations
match_stations_dict_list	Dictionary with stations that did match	match.csv	match_stations
decoded_stations_dict_list	Dictionary with match and no match stations found in the sample file	decoded.csv	decoded_stations
all_stations_dict_list	Dictionary with the list of official stations and lines from mapa_subte.jpg	Official.csv	official_stations

So, the last part of the process, having built the tables in the subway.db is to use a couple of queries to look for the best official station candidate name given the tokenized version that cames from pyton. The result , the final part, is a list where the wrong name of stations that did not match has their corresponding match name found.

Whislist

I would like to include a list of things -the ones I remembered- that can be improved and the ones that would be nice to have:

1) I included the line letter for each station . It could be used to create a k/v pair with that info to submit it to OSM.

2) I could investigate a less verbose procedure to try to match non-matching stations using python regex for examples

3) I could find a way to deal with XML big files (after sample them with DD) without caching exceptions

4) Work the unicode versions

5) Propose the Buenos Aires city government to have the data in a programmatic source (ej XML + web services) instead of just the jpg image.