Project 2: Data Wrangle OpenStreetMap Data

Mark Cassar

Map Area

For this project, I chose the map area of Cleveland, Ohio, USA. I decided on this area because it is a city that I frequently visited to see my cousins when I was younger, so am reasonably familiar with the area; it was also of sufficient size to get a good feel for the data wrangling process.

Link to city limits: *https://www.openstreetmap.org/relation/182130*
Link to XML OSM file: *http://metro.teczno.com/#cleveland*

Problems Encountered

I first carried out an exploration of the tags, keys, and structure of the Cleveland area map data via data_explore.py, which gave me a sense of the size and scope of elements to be found in the dataset. I then carried out a more systematic audit of the data (see audit_streets_zips_city.py) and discovered problems with the city streets, zip codes, and city names that seemed worthy of cleaning. As it was clear that the dataset included an area well beyond the city limits of Cleveland, and that this was a bounding box definition problem and not a data problem, I decided to treat the entire area in a consistent manner regardless of which neighboring cities were included. I now address each of the above-mentioned problems in turn.

Streets

A basic audit of the street names encountered in the map data showed two particular problems: abbreviated street names and missing street types.

All abbreviated street names were programmatically expanded to full words, so that “Erie Ave NW” became “Erie Avenue Northwest”. Care had to be taken to avoid creating problems in the process, such as expanding “Paul E. Brown Drive” to “Paul East Brown Drive”.

Missing street types were small in number so were fixed via a manually created list that was then used to update the data file. In this way, names like “Paula” and “South Arlington” were corrected to “Paula Drive” and “South Arlington Street”.

Zip Codes

A deeper look at the zip codes for this area showed that the main issues were due to non-numeric zip codes (e.g., “Ohio”) and inconsistent formats (some 5-digit format, e.g., “44129”, and others in 9-digit format, e.g., “44113-4466”). I chose to remove all non-numeric zip codes and to replace all the numeric ones with the 5-digit format.

Cities

The city information in the data showed a bit more variability in types of problems:

- Inconsistent use of abbreviations: corrected by expanding all abbreviations, so “Cleveland Hts.” became “Cleveland Heights”
- Spelling errors: “Heighst” was corrected to “Heights”
- Inappropriate inclusion of state in the city name: this I corrected by deleting the substring that included the state, so “Cleveland, OH” and “Cleveland OH” became “Cleveland”
- Incorrect entries: corrected by converting a city like “20770 Hilliard Blvd.” to the element’s address (consistent with method noted in **Steets** section above) and then adding in the appropriate city from a manually curated list

Data Overview

The main files used for this project and their sizes are in the following table.

File	Size
cleveland.osm	270 MB
cleveland.osm.json	294 MB

Some basic statistics of my chosen dataset and the MongoDB queries, through pymongo interface, used to gather them can be found in the following table.

Description	Query	Result
Number of documents	db.cleveland.find().count()	1,279,300
Number of nodes	db.cleveland.find({"type":"node"}).count()	1,164,751
Number of ways	db.cleveland.find({"type":"way"}).count()	1,114,534
Number of unique users	len(db.cleveland.distinct("created.user"))	554
Number of hospitals	db.cleveland.aggregate([{"$match": {"amenity":{"$exists":1}, "amenity":"hospital"}}, {"$group":{"_id":"$amenity", "count":{"$sum":1}}}])	124
Number of hospitals with emergency rooms	db.cleveland.aggregate([{"$match": {"amenity":{"$exists":1}, "amenity":"hospital", "emergency":"yes"}}, {"$group":{"_id":"$amenity", "count":{"$sum":1}}}])	10

Additional Ideas

The number of amenities in this dataset is certainly not an accurate representation of reality; while there may be 1656 schools, I doubt there is 1 doctor's office and 1 fitness center. This could be improved through a mechanism like the 'check-in' function that Foursquare used to have, or a partnership with such a company. In this way, data about locations could be collected for OpenStreetMap as a byproduct of people using another, perhaps much more popular, app.

A similar idea could be used for enhancing the the highway and street data by collaborating with a company like Waze. While the main goal of Waze is to be a community-based traffic and navigation app, there is enough gamification in the system to allow extra information to be collected about the roads people are already traveling and sharing information about.

I think it would also be helpful to include zip or postal codes as relations and then use these to automatically assign codes whenever an element's latitude and longitude falls within a given 'relation'. As such codes do not change frequently, this would reduce or eliminate the inconsistency and error rates found within the map data.

Another possibility that could help get more obscure paths into the map would be to enable the import of data from Google's My Tracks.

Conclusion

After reviewing the data and cleaning up the parts as noted, it is clear that there is a great deal more that should also be audited for this area. For example, the dataset included a good deal of information on water-based structures due to Cleveland being on the shore of Lake Erie. It is also clear that a lot of information has not been captured, particularly on the amenity side of things. It would seem most promising to explore its collection as a byproduct of people using some other GPS enabled smartphone app.

References

http://docs.mongodb.org/manual/
http://stackoverflow.com/
http://discussions.udacity.com/t/valueerror-importing-json-into-mongodb/21959
http://www.zip-codes.com/city/OH-CLEVELAND.asp
https://wiki.openstreetmap.org/