OpenStreetMap Data Case Study

Campbell, CA, USA

Aziz Mamatov

Data Source

I selected Campbell, CA as it is the city where I live. The data was downloaded from https://mapzen.com/ (https://mapzen.com/) and it had the information for the city already.

Data conversion and cleaning

XML file was large and I first selected the sample and converted it to CSV format. I then manipulated the data samples.

My final file sizes are:

File	Size
Downloaded xml file	77.5 MB
Sample (1/100) xml file	799 kB

Creating sample file

Sample file was created using sample.py. The following libraries were used and following name variables for sample and full size OSM files.

In [2]:
```
import xml.etree.ElementTree as ET
import pprint
import re
import collections
```

In [3]:
```
filename = 'campbell_mapzen.osm'
sample = 'campbell_map_sample2.osm'
```

Opening and parsing OSM document

Number of tags

I run the code tags.py and it revealed to have the following number of tags in full file of map : {'node': 333973, 1: 398199, 2: 234249, 3: 5782, 4: 1, 6: 50835, 8: 333973, 'nd': 398199, 'bounds': 1, 'member': 5781, 'tag': 234249, 'relation': 657, 'way': 50178, 'osm': 1})

The sample file have much less tags. defaultdict(<type 'int'>, {'node': 3339, 'nd': 4264, 'bounds': 1, 'member': 99, 'tag': 2441, 'relation': 7, 'way': 502, 'osm': 1})

Tag pattern information

I am checking information using `tags_patterns.py` contained within tags - 'k' values to see the patterns associated with tags. I selected regular expressions taken from Case studies. There are four tag categories:

- 'problemchars': 0, "problemchars", for tags with problematic characters
- 'lower': 1249, "lower", for tags that contain only lowercase letters and are valid
- 'other': 71, "other", for other tags that do not fall into the other three categories.
- 'lower_colon': 1121 "lower_colon", for otherwise valid tags with a colon in their names

Users

I will find out the number of unique users contributed to the map by producing a set of unique user IDs ('uid'). There are 646 users in the full file.

In [6]:
```
def get_user(element):
    return element.get('uid')


def process_map(filename):
    users = set()
    for _, element in ET.iterparse(full_filename):
        user = get_user(element)
        if user is not None:
            users.add(user)

    return len(users)

if __name__ == "__main__":
    print process_map(full_filename)
```

```
646
```

Street name wrangling

I was auding the xml file using `audit2.py` for 1) street names and 2) Zip codes and there were issues some of which were corrected.

There were peculiar names, but because they were so called borrowed words I left them as they are:

- 'Madrid': set(['Corte de Madrid']),
- 'Plaza': set(['Portal Plaza']),
- 'Sorrento': set(['Via Sorrento']),
- 'Barcelona': set(['Calle de Barcelona']),
- 'El Paseo': set(['El Paseo de Saratoga;])

However, there were legitimate mistakes, like:

- 'Ln' = Lane
- 'Rd' = Road
- 'Boulvevard' or 'Blvd' = Boulevard
- 'Ave' = Avenue
- 'St' = 'Street'

For zip code files - there was one zip code in map, which did not start with '95' but with '94'. 94 code belongs to Sunnyvale and for some reason was included to Campbell map.

Results of audit run:

```
defaultdict(<type 'set'>, {'Winchester': set(['Winchester']), 'Ln': set(['Branham Ln']), 'Rd': set(['Homestead Rd']),
'7.1': set(['Hwy 17 PM 7.1']), 'Hill': set(['Blossom Hill']), 'Circle': set(['Winchester Circle', 'Calabazas Creek
Circle', 'Greenwood Circle', 'Joseph Circle', 'Bobolink Circle']), 'Expressway': set(['Almaden Expressway',
'Southwest Expressway']), 'Alameda': set(['The Alameda']), 'Highway': set(['Monterey Highway']), 'Real': set(['El
Camino Real']), 'Boulvevard': set(['Los Gatos Boulvevard']), '1': set(['Prospect Rd #1']), 'Flores': set(['Terreno De
Flores']), 'Marino': set(['Via San Marino']), '6': set(['Pruneridge Ave #6']), 'Volante': set(['Via Volante']), 'Bascom':
set(['S. Bascom']), 'Dr': set(['Samaritan Dr', 'Linwood Dr']), 'Esquela': set(['Camina Esquela']), 'Bellomy':
set(['Bellomy']), 'Napoli': set(['Via Napoli']), 'Saratoga': set(['El Paseo de Saratoga']), 'Plaza': set(['Portal Plaza']),
'Barcelona': set(['Calle de Barcelona']), 'St': set(['N 5th St', 'Monroe St']), 'Mall': set(['Franklin Mall']), 'Franklin':
set(['Franklin']), 'Palamos': set(['Via Palamos']), 'Seville': set(['Corte de Seville']), 'Presada': set(['Paseo
Presada']), 'Loop': set(['Infinite Loop']), '4A': set(['Saratoga Avenue Bldg 4A']), 'Sorrento': set(['Via Sorrento']),
'Portofino': set(['Via Portofino']), 'Julian': set(['West Julian']), 'Walk': set(['Paseo de San Antonio Walk']), 'Terrace':
set(['Hobart Terrace']), 'Madrid': set(['Corte de Madrid']), 'Blvd': set(['Stevens Creek Blvd', 'Los Gatos Blvd']),
'Ave': set(['Cherry Ave', 'Greenbriar Ave', 'Blake Ave', 'Foxworthy Ave', 'Meridian Ave', 'Westfield Ave', 'The
Alameda Ave', 'N Blaney Ave']), 'Row': set(['Santana Row'])})
```

Results of corrected street names and codes (sample):

- Winchester => Winchester
- Branham Ln => Branham Lane
- Homestead Rd => Homestead Road
- Hwy 17 PM 7.1 => Hwy 17 PM 7.1
- Blossom Hill => Blossom Hill
- Almaden Expressway => Almaden Expressway
- Southwest Expressway => Southwest Expressway
- The Alameda => The Alameda
- El Camino Real => El Camino Real

Zip codes audit

Zip codes were audited using `zip_codes_audit.py` file and no discrepancies were found.

CSV file preparation

Now I will save my XML file to a CSV using `data_csv2.py`. Udacity provided with a starter code that where I added shape_element function. As a result, the following files were prepared from sample file:

File name	Size
nodes.csv	281KB
nodes_tags.csv	11KB
ways.csv	100 KB
ways_nodes.csv	30KB
ways_tags.csv	74KB

Then, I repeated the process on the full file and the files with following size were created:

File name	Size
nodes.csv	28.1MB
nodes_tags.csv	1.1MB
ways.csv	3MB
ways_nodes.csv	9.4MB
ways_tags.csv	7MB

Preparing sql database

I prepared sql database file campbell_map2.db (55.3MB) using Mac shell. https://discussions.udacity.com/t/creating-db-file-from-csv-files-with-non-ascii-unicode-characters/174958/2 (https://discussions.udacity.com/t/creating-db-file-from-csv-files-with-non-ascii-unicode-characters/174958/2)

- nodes: "id", "lat", "lon", "user", "uid" , "version" , "changeset" , "timestamp"
- nodes_tags: "id" , "key" , "value" , "type"
- ways: id" , "user" , "uid" , "version" , "changeset", "timestamp"
- ways_nodes: "id" , "node_id" , "position"
- ways_tags: "id" , "key" , "value" , "type"

Some of the dataset statistics using sql queries

Top 15 tags used overall

QUERY = "SELECT Allkeys.key, COUNT(key) FROM (SELECT *FROM nodes_tags UNION ALL SELECT* FROM ways_tags) Allkeys GROUP BY Allkeys.key ORDER BY COUNT (key) DESC LIMIT 15;"

- building,26724
- highway,23688
- name,16575
- county,11073
- name_base,9953
- name_type,9543
- zip_left,8901
- cfcc,8794
- zip_right,8623
- street,7845
- housenumber,7740
- postcode,6420
- height,5973
- oneway,4880
- source,4657

Total unique users

QUERY = "SELECT COUNT(DISTINCT(a.user)) FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) a;"

640

Number of types in ways_tags column

QUERY: SELECT type, COUNT(*) as num

- FROM ways_tags
- GROUP BY type
- ORDER BY num DESC
- LIMIT 4;
    - regular,104965
    - tiger,75151
    - addr,15719
    - turn,2248

Similar story is with nodes_tages file. An absolute majority of tags were marked as 'regular' as they did not have any value assigned to them.

Number of nodes and ways

QUERY = select count(distinct(id)) from nodes;

- 333973

QUERY = select count(distinct(id)) from ways;

- 50178

Top users

QUERY = "SELECT B.user, COUNT(*) as num

- FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) B
- GROUP BY B.user
- ORDER BY num DESC
- LIMIT 10;

- dannykath,40066
- "Bike Mapper",39716
- nmixter,39534
- karitotp,34245
- mk408,20125
- "Minh Nguyen",17964
- samely,15114
- doug_sfba,14559
- beddy,11635
- RichRico,10413

Top amenities in nodes_tags and ways_tags table

These two tables had to be combined in order to pick all the values for certain amenities QUERY = SELECT value, COUNT(*) as num

- SELECT DISTINCT(a.value), COUNT(*) as num
- FROM (SELECT *FROM nodes_tags UNION ALL SELECT* FROM ways_tags) a
- WHERE key='amenity'
- GROUP BY a.value
- ORDER BY num DESC
- LIMIT 10;

- parking,767
- restaurant,398
- school,247
- place_of_worship,212
- fast_food,206
- bench,120
- cafe,106
- fuel,87
- bicycle_parking,84

Rare cuisines

- SELECT DISTINCT(a.value), COUNT(*) as num
- FROM (SELECT *FROM nodes_tags UNION ALL SELECT* FROM ways_tags) a
- WHERE key='cuisine'
- GROUP BY a.value
- ORDER BY num ASC
- LIMIT 10;
  - Coffee,1
  - Coffee_shop,1
  - Diner,1
  - Sandwich,1
  - american;burger,1
  - bagels,1
  - bakery,1
  - boba,1
  - boba_&_snacks,1
  - boba_tea,1

As it can be seen some of the cuisines are not classified in the best manner - I would probably classify Diner as American as well as Bagels etc.

Conclusion and Problems with data_sets

The OpenStreetMap data of Cambell is of a good quality but there are some typo errors as well as classification mistakes. I did not have to clean a lot of data, as most probably such cleaning was already performed in the past. Data contains lots of information about amenities but it is not clear whether it is updated regularly. This leads to the main theory behind the open map project - are there are enough diligent contributors to the map?

As we can see from sql query the top 10 contributors are pretty close to one another in contributions and it means that there is a good chance of igniting competition. It can be done via gamification. Gamification can be obtained via publicizing the competition via social networks (FB, twitter, reddit) and actually awarding certain distinctions to top contributors overall, by continent, country, state, city etc. Top contributors would be awarded certain ranks.

Pros:

It will motivate contributors to contribute even more and attact new contributors.

Cons:

Somebody will need to moderate and track the process and it may take significant time. There also may be controversies

Also, a good way of increasing engagement with the openmaps is to create a user friendly apps in iOS and Android that would allow contributions.

Pros:

Apps will possibly have higher engagements as they are easier to be worked with at any time contributor feels like updating or correcting the map.

Cons:

Somebody will have to create the apps and maintain them. Also, somebody will have to monitor the increased contributions to the map.

Some technical issues.

- Resulting CSV files had repeating IDs for each tag - it would be a good idea to expand the tables by adding columns and have each UID for each row.
- Some of the data in openmaps, like classification of restaurants by cuisine was not always done in the best manner - there must be some general classification of cuisines in order not to have too many of them.
- An absolute majority of tags were marked as 'regular' during data wrangling as they did not have any value assigned to them. It would be a good practice to either programmatically assign value to them or prompt the contributors to assign value.

Sources:

https://discussions.udacity.com/t/no-user-and-uid-in-a-node/170245/7 (https://discussions.udacity.com/t/no-user-and-uid-in-a-node/170245/7) https://www.tutorialspoint.com/sqlite/sqlite_using_joins.htm (https://www.tutorialspoint.com/sqlite/sqlite_using_joins.htm)
https://gist.github.com/swwelch/f1144229848b407e0a5d13fcb7fbbd6f (https://gist.github.com/swwelch/f1144229848b407e0a5d13fcb7fbbd6f)
http://stackoverflow.com/questions/8331469/python-dictionary-to-csv (http://stackoverflow.com/questions/8331469/python-dictionary-to-csv)
http://stackoverflow.com/questions/15707056/get-time-of-execution-of-a-block-of-code-in-python-2-7 (http://stackoverflow.com/questions/15707056/get-time-of-execution-of-a-block-of-code-in-python-2-7)
https://wiki.openstreetmap.org/wiki/Category:Features (https://wiki.openstreetmap.org/wiki/Category:Features)
https://discussions.udacity.com/c/nd002-p3-data-wrangling (https://discussions.udacity.com/c/nd002-p3-data-wrangling) http://www.sqlabs.com/blog/2010/12/getting-the-size-of-an-sqlite-database/ (http://www.sqlabs.com/blog/2010/12/getting-the-size-of-an-sqlite-database/)
http://scottboms.com/downloads/documentation/markdown_cheatsheet.pdf (http://scottboms.com/downloads/documentation/markdown_cheatsheet.pdf)

Files

- `audit2.py`: audit names of streets of the city
- `campbell_map_sample_2.osm`: sample file of osm data file
- `data_csv2.py`: build CSV files from OSM and clean data
- `P3_OpenStreetMap_CaseStudy2.pdf`: report file
- `sample.py`: sample data extraction code
- `sql_queries_project`: SQLITE queries
- `tags.py`: tags code
- `tags_patterns.py`: tags patterns code
- `campbell_map2`: SQLITE database file
- `nodes_tags.csv`: csv file
- `nodes.csv`: csv file
- `ways_nodes.csv`: csv file
- `ways_tags.csv`: csv file
- `zip_codes_audt.py`: audit file for zip codes
- `campbell_mapzen.osm`: full OpenMap extraction file

```
In [ ]:
```