OpenStreetMap Project
Data Wrangling with MongoDB

Meilan Ou

Map area: Hong Kong SAR (HK), China
OSM data source: https://s3.amazonaws.com/metro-extracts.mapzen.com/hong-kong_china.osm.bz2
Chungking Mansion: https://www.openstreetmap.org/export#map=17/22.29638/114.17193&layers=C

1. Overview of the data

This section contains basic statistics about the dataset and the PyMongo queries used to gather them. For detailed codes, please see 2. Overview of the data in IPython Notebook.

File sze

After extraction, the OSM XML file hong-kong_china.osm has about 411 MB in size.

Number of documents

To preserve as much original data as possible, I imported OSM elements (nodes, ways, relations and respective sub-elements) into db.elems in MongoDB all together. Also I aggregated unique users' contribution in element counts into db.users. Below I'm checking the counts of each type and verifying the total counts from db.elems and db.users collections.

```
def check_counts():
    pp.pprint("Count of all elements: {0}".format(db[COLLECTION_ELEMS].count()))
    pp.pprint("Count of {0} elements: {1}".format(TAG_NODE, db[COLLECTION_ELEMS].count({"type": TAG_NODE})))
    pp.pprint("Count of {0} elements: {1}".format(TAG_WAY, db[COLLECTION_ELEMS].count({"type": TAG_WAY})))
    pp.pprint("Count of {0} elements: {1}".format(TAG_RELATION, db[COLLECTION_ELEMS].count({"type": TAG_RELATION})))
    pp.pprint("Count of unique {0}: {1}".format(COLLECTION_USERS, db[COLLECTION_USERS].count()))
    pipeline = [{"$group": {"_id": "mock", "totalCount": {"$sum": "$count"}}}]
    pp.pprint("Count of all elements by users: {0}".format(db[COLLECTION_USERS].aggregate(pipeline).next()["totalCount"]))
```

Count of all elements: 2239509
Count of **node** elements: 2035874
Count of **way** elements: 198007
Count of **relation** elements: 5628
Count of **unique users**: 1155
Count of all elements by users: 2239509

MongoDB collection size

Although not directly providede by PyMongo, MongoDB has the collStats command returns a variety of storage statistics for a given collection. Below I'm running this command to see how big the major collection “db.elems” is after importing into MongoDB.

```
def init_database(loc, name):
    ……
    for col in collections:
        print "Collection {}:".format(col)
        mypp.pprint(db.command("collstats", col))
```

Collection elems:
{……
 size: 656761520, ---> about 626.34 MB
 ……}

Top map features

Because OSM map features were entered by contributors. It would be interesting to see what features are more important for this map area. Besides features about infrastructure, obviously you can find English and Chinese names are both used in HK. That is aligned with the local custom of this multi-linqual area.

```
def check_top_features():
    limit = 10
    for elem_type in ELEMENTS_CORE:
        pipeline = [{"$match": {"type": elem_type, "tags": {"$exists": 1}}},
                    {"$project": {"tags": 1}},
                    {"$unwind": "$tags"},
                    {"$group": {"_id": "$tags.k", "count": {"$sum": 1}}},
                    {"$sort": {"count": -1}},
                    {"$limit": limit}]
        features = [i for i in db[COLLECTION_ELEMS].aggregate(pipeline)]
        print "Top {0} features of {1} elements.".format(limit, elem_type)
        mypp.pprint(features)
```

Top 10 features of node elements.
[{_id: name, count: 20680},
 {_id: highway, count: 14839},
 {_id: power, count: 11035},
 {_id: name:en, count: 8719},
 {_id: name:zh, count: 8625},
 ……]
Top 10 features of way elements.
[{_id: highway, count: 119355},
 {_id: name, count: 50726},
 {_id: oneway, count: 43762},
 {_id: name:zh, count: 34075},
 {_id: name:en, count: 33461},
 ……]
Top 10 features of relation elements.
[{_id: type, count: 5400},
 ……

{_id: name:en, count: 351},
{_id: operator, count: 335},
{_id: name:zh, count: 308}]

Top contributing user

As of April 2nd 2015, there are 1155 unique users who had edited OSM of this area. Out of 7.2 millions population, about 0.016% are contributing (just a reference as OSM can be edited anywhere). HK already has more data than other big cities like Beijing, Shanghai, Singapore and etc. OpenStreetMap may allocate more time to promote in Asia. In HK, top 10% users contributed to 96.52% of the map data.

```
def get_count_by_top_contributors(limit, display=False):
    pipeline = [{"$group": {"_id": "$_id", "totalCount": {"$sum": "$count"}}},
                {"$sort": {"totalCount": -1}},
                {"$limit": limit}]
    contributors = [i for i in db[COLLECTION_USERS].aggregate(pipeline)]

    if display:
        print "Top {0} contributors:".format(limit)
        mypp.pprint(contributors)

    topTotalCount = 0
    for i in contributors:
        topTotalCount += int(i["totalCount"])

    return topTotalCount
```

Total count of unique contributors: 1155
Average element count per contributor: 1938.97
Top 12 contributors:
[{_id: {uid: 169827, user: hlaw}, totalCount: 522234},
 {_id: {uid: 44514, user: MarsmanRom}, totalCount: 200635},
 {_id: {uid: 27454, user: Popolon}, totalCount: 165927},
 ……]
Contribution of top 1% contributors: 65.45%
Contribution of top 10% contributors: 96.52%

Chosen type of nodes, ways and relations: tourism

Besides residential and business uses, HK is also a tourism destination. Out or my curiosity, I'm checking what have been logged for tourism. Interestingly, guest_house shows up on the top. HK is well known for being expensive, but there are also cheap accommodation alternatives for backpackers such as guesthouses in Chungking Mansion. This leads to the next section of data exploration.

```
def check_top_feature_values(elemTypes, key, limit):
    pipeline = [{"$match": {"type": {"$in": elemTypes}, "tags.k": key}},
                {"$project": {"tags": 1}},
                {"$unwind": "$tags"},
                {"$match": {"tags.k": key}},
                {"$group": {"_id": "mock", "count": {"$sum": 1}}}]
```

```
featureCount = db[COLLECTION_ELEMS].aggregate(pipeline).next()["count"]
print "{0} feature of {1} were logged".format(featureCount, key)

pipeline = [{"$match": {"type": {"$in": elemTypes}, "tags.k": key}},
            {"$project": {"tags": 1}},
            {"$unwind": "$tags"},
            {"$match": {"tags.k": key}},
            {"$group": {"_id": "$tags.v", "count": {"$sum": 1}}},
            {"$sort": {"count": -1}},
            {"$limit": limit}]
values = [i for i in db[COLLECTION_ELEMS].aggregate(pipeline)]
print "Top {0} values of {1}:".format(limit, key)
mypp.pprint(values)
```

```
2852 feature tags of tourism were logged
Top 10 values of tourism:
[{_id: information, count: 1248},
 {_id: attraction, count: 481},
 {_id: hotel, count: 449},
 {_id: guest_house, count: 149},
 {_id: viewpoint, count: 131},
 {_id: hostel, count: 89},
 {_id: picnic_site, count: 84},
 {_id: museum, count: 79},
 {_id: artwork, count: 63},
 {_id: camp_site, count: 49}]
```

2. Data exploration (and auditing) of Chungking Mansion

Chungking Mansions is well known to tourists as nearly the cheapest accommodation in HK. Though the building is supposedly residential, it is made up of many independent low-budget hotels, shops and other services. The unusual atmosphere of the building is sometimes compared to that of the former Kowloon Walled City (http://en.wikipedia.org/wiki/Chungking_Mansions).

On the map it seems someone was logging the cheap hotels (guesthouses) in the complex. Below I'll exploring those programmatically. For detailed codes, please see 3. Data exploration in IPython Notebook.

Custom feature tags

In OSM, Chungking Mansion is a “way” elements consisted of 7 nd sub-elements that represent a closed polygon on the map. It's awesome that MongoDB has geo functions built int so that I could easily find all the hotels/guesthouses located withing this polygon. And there are 56 hotels/guesthouses found. Out of these 56 hotels, 16 span across multiple unit horizontally or vertically, but not necessarily adjacent units.

I noticed that contributors are accustomed to use this format to log the apartment unit {k: addr:door, v: 9E5}. 9E5 stands for: room #5, 9th floor, block E (Chungking Mansion is consisted of 5 blocks A, B, C, D and E). Sometimes if the hotel spans cross multiple units, we will see an element like {k: addr:door, v: 8B1-3} or {k: addr:door, v: 8B1,3,5}. And “addr:door” does not always present.

Such element constantly comes with these tags: {k: level, v: 9}
{k: addr:floor, v: 9/F}
{k: addr:street, v: 彌敦道 Nathan Road}
{k: addr:housenumber, v: Block E, 36-44}

```
def explore_chungking():
    building = db[COLLECTION_ELEMS].find_one({"tags.k": "name",
                                    "tags.v": {"$in": ["Chungking Mansion", "Chungking Mansions", "重慶大廈"]}})
    ……
    cursor = db[COLLECTION_ELEMS].find({"type": "node",
                                "tags.k": "tourism",
                                "tags.v": {"$in": ["apartment", "guest_house", "hostel", "hotel", "motel"]},
                                "loc": {"$geoWithin": {"$polygon": polygon}}})
    ……
    return hotels
```

Chungking Mansion has a total of 56 hotels
40 hotels with usual door numbers
16 hotels with missing/unusual door numbers

3. Problems encountered in the map

$Unwind

At first I didn't extract address sub-elements to preserve as much original data as possible. But when I started exploring hotels in Chungking Mansion I found that querying a list of objects required more work. And I needed to use $unwind constantly to aggregate the correct data. So, for future convenience in the fixing section I'll create new dictionary item for address and update these documents in MongoDB.

Custom feature tags

As mentioned previously, contributors are accustomed to use this format to log the apartment unit {k: addr:door, v: 9E5}. 9E5 stands for: room #5, 9th floor, block E. But this can be misread easily. Also, for {k: addr:floor, v: 9/F}, suffix “/F” is redundant and can add more work for data processing. I did a quick check of these custom feature tags below. There are special cases for addr:door and addr:floor, and some missing tags. Otherwise, the data are pretty consistent.

```
def check_addr_chungking(hotels):
    addr_stats = {"addr:housenumber": {},
              "level_floor": {},
              "addr:door_special": []}
    ……
```

```
{'addr:door_special': [15B1,6,8,
                       8B1-3,
                       ……
                       'missing(housenumber=Block A, 36-44, level=4, floor=4)'],
```

'addr:housenumber': {Block A, 36-44: 12,
Block B, 36-44: 16,
Block C, 36-44: 8,
Block D, 36-44: 14,
Block E, 36-44: 5,
'missing(door=7D8)': 1},
'level_floor': {'10_10': 2,
…...
'4_4': 4,
'4_4,5': 1,
…...
'7_missing(door=7D8)': 1,
…...}}

4. Data correction of Chungking Mansion

For detailed codes, please see 4. Data correcting and 5. Data exploration - after correction in IPython Notebook.

Fixing the data

The goal is to extract values from tag elements and add as new dictionary items for quick access. The new format will be:

name: Comfort Guesthouse,
addr: {block: E,
floor: 9,
housenumber: Block E, 36-44,
room: 5,
street: 彌敦道 Nathan Road}

In HK "room" is used to mark apartment units in a residential building. Although in OSM "room" is used to mark the purpose of a space in the building.

The contributor used "addr:door" for apartment units and I'll extract it to addr-room and make it more harmonized (9E5 -> 5, 15B1,6,8 -> 1,6,8). And contributors like to put block info with addr:housenumber. Since "block" is commonly used in HK, I'll extract block and put it in addr-block (Block A -> A).

Contributors use suffix "/F" for addr:floor tags and it's redundant. I'll remove "/F" and just use numbers.

If a guesthouse spans vertically, since I don't know how it's configured I'll use multiple values for addr-floor (4/F-5/F -> 4,5).

```
def correct_chungking():
    ……
    matched_count = 0
    modified_count = 0
    for d in update_docs:
        result = db[COLLECTION_ELEMS].update_one(d["filter"], d["update"], upsert=True)
        matched_count += result.matched_count
        modified_count += result.modified_count
    ……
```

56 documents prepared for update. 56 found for update, and 56 updated.

Verifying after correction

Now we can extract the directory of all the hotels in Chungking Mansion with much less code due to harmonized data. And one example of the new format:

```
{_id: ObjectId('552f47c8e3f353197837c26c'),
 addr: {block: E,
        floor: 9,
        housenumber: Block E, 36-44,
        room: 5,
        street: □ □ □  Nathan Road},
 created: {changeset: 23262762,
           timestamp: 2014-06-29T05:25:45Z,
           uid: 474022,
           user: sashazykov,
           version: 3},
 id: 2935557575,
 loc: [114.1730717, 22.2961491],
 name: Comfort Guesthouse,
 tags: [{k: name, v: Comfort Guesthouse},
        {k: level, v: 9},
        {k: tourism, v: guest_house},
        {k: addr:door, v: 9E5},
        {k: addr:floor, v: 9/F},
        {k: addr:street, v: □ □ □  Nathan Road},
        {k: addr:housenumber, v: Block E, 36-44}],
 type: node}
```

5. Other ideas and conclusion

If you compare Chungking Mansion on OSM and Google Map, you will realize that you can't see the density of guesthouses on Google Map. But the high concentration had drawn researchers' attention to spend 4 years to just study this residential complex (http://www.amazon.com/Ghetto-Center-World-Chungking-Mansions/dp/0226510204). Co-creation can be really powerful especially on circumstances that are only known by the local crowds. However as mentioned earlier in HK only about 0.016% of the population are contributing to OSM. And take Chungking Mansion for example, only 56 out of 89 listed guesthouses were logged. Lots of businesses are still missing on OSM.

I tried logging the complex I currently live in but feel that creating a new node is not a very efficient process. Maybe a GPS app for walking (other slow speed) activities with edge recognition and gamification built in can encourage people to log more nodes on the go.

OSM data worth do much as not only what's logged but also why it's logged by the crowd means so much more than just business directories.