OpenStreetMap Project
Data Wrangling with MongoDB

Gabor Sar

February 25, 2015

Map area: Budapest, Hungary
https://mapzen.com/metro-extracts/

1 Problems Encountered in the Map

After I downloaded the data and I had an initial read on it I immediately saw a few errors in the address fields and the keys. In order to get a better picture about the quality of the data and those issues, I generated files containing lists of unique values of the problematic fields and the keys (Listing 1, *unique_values.py*). Based on those unique values I have defined the validation rules I have used (*validation.py*). Using those validation rules I ran a basic audit of the data (Listing 1, *audit.py*). After the initial audit, I started to define the cleaning logic (*clean.py*), and I kept improving it until the audit stopped reporting invalid values.

To audit postcodes, city names and street names I used a publicly available reference data [2], which I downloaded from the website of the Hungarian Post Office. To make it easier to use, I transformed the data into a JSON file (*standard_data.json*).

In the process of auditing the data, I found the following problems:

- Invalid keys
- Inconsistent keys
- Typographical errors in keys
- Inconsistent city names
- Typographical errors in city names
- Inconsistent house number formats
- Topographical numbers instead of house numbers
- Invalid postcodes
- Inconsistent state formats
- Invalid city names
- Topographical and house numbers in street names
- Typographical errors in street names

Listing 1: Generate lists of unique values of the problematic fields and the keys and audit the data.

```
$ python unique_values.py
$ python audit.py
```

1.1 Keys

Three keys contained dots instead of colons to separate different levels of groups (e.g., *"surface.material"* instead of *"surface:material"*).

In a few cases, more than one key was used to represent the same type of information (e.g., *"Street"* instead of *"addr:street"*).

I also found typographical errors (e.g., *"acess"* instead of *"access"*, *"disusedhighway"* instead of *"disused:highway"*).

I replaced all the dots with colons and the invalid keys with their valid versions.

1.2 City Names

In some cases where the city contained the name of a part of the city, it did not contain the name of the city itself (e.g., *"Agárd"* instead of *"Gárdony-Agárd"*).

I found a few typographical errors as well (e.g., *"Agáed"* instead of *"Agárd"*).

I replaced all the invalid city names with their valid versions.

1.3 House Numbers

There were many different house number formats used in the data. I selected the most commonly used format as the standard format and using regular expressions I transformed most of the house numbers to their valid format.

The valid house number consist of one or more units (numbers and alphabets separated by slashes - house/entrance/level/flat) connected with dashes (range) or separated by commas (list).

After the standardization, only a short list of invalid values remained (4 unique values). One of them was valid (*"19 km pihenő"*) and three of them were topographical numbers (*"019/8 hrsz"*, *"081/15 hrsz"*, *"15/7 hrsz"*). I updated the key of the topographical numbers to the appropriate *addr:hrsz*.

1.4 Postcodes

The data contained both Hungarian and Slovakian addresses and therefore, postcodes.

A valid Slovakian postcode consists of 5 digits with a space after the third digit [1]. Two of the

Slovakian postcodes did not contain the space (e.g., *"94301"* instead of *"943 01"*).

Based on the reference data [2], some of the Hungarian postcodes were invalid. After I have checked the actual addresses, I understood most of them were postcodes of post offices, museums, supermarkets and other private businesses. Those tend to have their special postcodes. Still, four of them were invalid due to possible typographical errors (e.g., *"1231"* instead of *"1213"*).

I added the missing space to the invalid Slovakian postcodes and replaced all the invalid Hungarian postcodes with their valid versions.

1.5 States

All the addresses in the data had the same state value but in some cases it was uppercase instead of title case (*"CENTRAL HUNGARY"* instead of *"Central Hungary"*). I converted all the state values to title case.

1.6 Street Names

Auditing street names and street types revealed three types of issues.

I found invalid street names that I was not able to fix due to the lack of information about the particular address (Listing 2).

In other cases street names contained house or topographical numbers (e.g., *"Szent István körút 13"*, *"Liszenkó telep 0318 hrsz."*).

I also found typographical errors (e.g., *"uca"* instead of *"utca"*).

I replaced all the invalid street names with their valid versions, and I removed and reinserted the house and typographical numbers with their appropriate keys (*addr:housenumber*, *addr:hrsz*).

Listing 2: A non fixable street name issue.

```
<node id="1372162022" version="2">
  <tag k="addr:street" v="ny"/>
  <tag k="addr:housenumber" v
     ="125"/>
</node>
```

2 Data Overview

The following section provides a statistical overview of the dataset, as well as the used MongoDB queries.

The size of *budapest_hungray.osm* is 511MB, the size of *budapest_hungray.json* is 562MB, the number of *documents* is 2570746, the number of *nodes* is 2203666, the number of *ways* is 366985, the number of *unique contributing users* is 1721 and the top three *contributing users* are *igor2* (392265 contributions), *Separis* (152763 contributions) and *nagy_balint* (127935 contributions).

Listing 3 shows how to generate *budapest_hungray.json* from *budapest_hungray.osm* and store it in MongoDB, Table 1 shows the size of the two files and Listings 4 - 8 shows the MongoDB queries and the results of them.

Listing 3: "Generate *budapest_hungray.json* from *budapest_hungray.osm* and import it into MongoDB."

```
$ python data.py
$ mongoimport \
    --db openstreetmap \
    --collection hun \
    --file budapest_hungary.json
```

Table 1: File Sizes

budapest_hungray.osm	511MB
budapest_hungray.json	562MB

Listing 4: Number of documents.

```
> db.hun.count()
2570746
```

Listing 5: Number of nodes.

```
> db.hun.count({ type: 'node' })
2203666
```

Listing 6: Number of ways.

```
> db.hun.count({ type: 'way' })
366985
```

Listing 7: Number of unique contributing users.

```
> db.hun.distinct('created.user').
   length
1721
```

Listing 8: Top three contributing users.

```
> db.hun.aggregate([
.   { $group: {
.     _id: '$created.user',
.     contributions: { $sum: 1 }
.   } },
.   { $sort: {
.     contributions: -1
.   } },
.   { $limit: 3 },
.   { $project: {
.     user: '$_id',
.     _id: 0,
.     contributions: 1
.   } }
. ])
{
  "user": "igor2",
  "contributions": 392265
}
{
  "user": "Separis",
  "contributions": 152763
}
```

```
{
  "user": "nagy_balint",
  "contributions": 127935
}
```

3 Additional Ideas

After I generated (Listing 9) and compared (Table 2) the frequency of the different address keys, I noticed that the high-level keys (*postcode*, *city*, *country*) are missing in so many cases (only 48% of the addresses have a *country* value) as it is one of if not the most important improvement we can make.
One other area that we could improve is the lack of districts in addresses. Budapest has 23 districts and as they have a very important role in terms of governance of the city, adding them to our database would be a very good improvement.

3.1 Address Key Frequencies

A possible way to solve this issue is improving the incomplete addresses based on their neighbors. We can search other points in the data with a matching low-level address key (e.g., *street*) in a maximum distance (e.g., 100m), and we can copy the missing high-level keys from those points. We can repeat this process until we cannot find more points that we can improve.

3.2 Districts in Budapest

We can add districts to our database using the standard data we have [2]. We can assign district values to addresses based on *city*, *street* and *postcode*. We can also use the data to provide us the list of possible districts, where it is not possible to choose one automatically.

Listing 9: Get the frequency of the address keys.

```
> db.hun.mapReduce(
.   function map() {
.     var keys = Object.keys(this.
    address);
.     keys.forEach(function (key) {
.       emit(key, 1)
.     });
.     emit('address', 1);
.   },
.   function reduce(key, values) {
.     return Array.sum(values);
.   },
.   {
.     query: {
.       address: { $exists: true }
.     },
.     out: { inline: 1 }
.   }
. );
```

Table 2: Frequency of the address keys.

address	55262	100%
address.street	45150	82%
address.housenumber	40686	74%
address.postcode	33748	61%
address.city	32593	60%
address.country	26286	48%
address.interpolation	3162	6%
address.suburb	1864	3%
address.housename	371	1%
address.inclusion	139	<1%
address.staircase	48	<1%
address.conscriptionnumber	37	<1%
address.place	20	<1%
address.hrsz	12	<1%
address.state	11	<1%
address.unit	8	<1%
address.full	6	<1%
address.door	5	<1%
address.floor	5	<1%
address.source	3	<1%
address.conscriptionnumber_1	1	<1%
address.conscriptionnumber_2	1	<1%
address.district	1	<1%
address.housenumber_1	1	<1%
address.old_street	1	<1%
address.street_1	1	<1%

References

[1] Slovenská pošta. *Slovakian Postcodes*. URL: http://psc.posta.sk/en.

[2] Magyar Posta Zrt. *Hungarian Postcodes*. URL: http://posta.hu/static/internet/download/Iranyitoszam_Internet.XLS.