Open Street Map Project

Data Wrangling With MongoDB

Richard Haughton

References

1) shape.txt - https://github.com/studyrah/p2-datawrangling/blob/master/ud032/Lesson_6_OSM/project/docs/shape.txt
2) jsonconversion1.txt - https://github.com/studyrah/p2-datawrangling/blob/master/ud032/Lesson_6_OSM/project/docs/jsonconversion1.txt
3) streets.txt - https://github.com/studyrah/p2-datawrangling/blob/master/ud032/Lesson_6_OSM/project/docs/streets.txt
4) OSM Wiki Features - http://wiki.openstreetmap.org/wiki/Map_Features
5) Taginfo - http://taginfo.openstreetmap.org/download
6) tags.txt - https://github.com/studyrah/p2-datawrangling/blob/master/ud032/Lesson_6_OSM/project/docs/tags.txt
7) wikitagaudit.txt - https://github.com/studyrah/p2-datawrangling/blob/master/ud032/Lesson_6_OSM/project/src/wikitagaudit.py
8) wa-trunc.json – https://github.com/studyrah/p2-datawrangling/blob/master/ud032/Lesson_6_OSM/project/data/wa-trunc.json

   (note this is truncated to just 10000 lines, as the full size file is too large to upload)

Dataset Overview

My chosen dataset covers a region including Cardiff, Newport, Bristol and Bath in the UK

Below are some basic facts and figures about the dataset, a more detailed overview (including MongoDB queries as appropriate) is contained in [ref: 1]

File Size – 436MB (uncompressed)

Number of Lines – 6,173,090 (though not very illuminating as it's xml)

Tag Counts

{'bounds': 1, 'member': 44811, 'nd': 2585692, 'node': 2093098, 'osm': 1,

'relation': 1766, 'tag': 849051, 'way': 260051}

Key Character Validation

{'lower': 716267, 'lower_colon': 89936, 'other': 42838, 'problemchars': 10}

Distinct Contributors – 1711

Number of Records – 2,353,149

(explicitly this is the number of records to load into Mongo based on nodes/ways)

Problems Encountered

I concentrated on 3 different aspects of the data

- Street Types (as per lesson 6)
- Postcodes
- Tag Consistency

Each and the specific problems I looked to audit and/or address (e.g. tag key or value standardisation, postcode validity) are covered in the following sections.

Additionally I faced technical issues such as:

- efficiency of xml parsing for large datasets - for which I changed xml library
- date format parsing - in order to load dates as dates into Mongo.

Details in [ref: 2]

Street Type Audit

As per 'problem set 6' I performed an audit of street types.

In total I identified:

- **Valid street types** – 76 (e.g. Close, Gardens, View...)
- **Mappable variants** - 11 (e.g. Cresent to Crescent)
- **Other** – 157

There are many distinct 'Other' cases, most look like valid 'edge cases' and in total account for a very small percentage of the data.

In summary the street data for this dataset is already quite clean (suggesting it is mostly automatically input) but there are some gains from applying the identified mappings.

A more detailed summary can be seen in [ref: 3]

Post Codes Audit

Associated MongoDB queries for this section are contained in Appendix A – Postcode Queries

Tag Name Consistency

The OSM wiki [ref: 4] defines 'addr:postcode' as the best practice form. Using a map/reduce job over MongoDb I output all the tag keys with a count for each. From this it was trivial to extract all the 'likely' post code tag name variants.

`postal_code` `(6567) and` **`addr:postcode`** `(5947) dominated. For 'other variants' I decided to observe the data with MongoDB console queries, finding the following:`

field	count	example (where necessary)	explanation
source:postcode	156	OS_OpenData_CodePointOpen	This field (unsurprisingly) refers to the source that provided the postcode, so this is not a postcode field variant but a discreet piece of information
uk_postcode_centroid	79	-	an explanation of postcode centroids can be found here - http://codepoint.raggedred.net/ in particular though it says: ...PLEASE do not add the centroids to the OSM map database. The centroids are not real objects so they do not belong in the OSM database.
old_uk_postcode_centroid	10	-	see above
note:postal_code	3	-	unsurprisingly these are freetext comments regarding an associated postal_code field in the record
note:postcode	3	-	see above
postcode	1	-	Simply an alternative postcode field, incidentally all three examples are valid

Assuming we want to map postal_code and addr:postcode to the same thing, its worth checking how often we see both in the same record

Just 24 records contain both and in all but one case exactly match

Postcode Validity

I Conducted postcode validation according to a complex regex (Appendix A – Postcode Queries)

In particular I considered both popular cases (postal_code, addr:postcode)

The table below shows results of a series of queries to inspect the validity of each field respectively

- **How many (of each variant of postcode) are there?**

  To double check against the map/reduce results earlier

- **How many are valid UK postcodes?**

- **How many are invalid UK postcodes?** (for completeness – valid + invalid = 100%)

- **How many are valid 'partial postcodes'?**

  Inspection revealed that we often see partial postcodes, presumably denoting 'general area' ?

  Explicitly, uk postcodes are typically in 2 halves, the partial postcodes we see are either just the first half or the first half plus the first digit of the second half

  e.g. BS7, CF72 9

If we assume this is a valid thing to enter, how many are actually '**valid partial postcodes**'?

- **What are the remaining invalid postcodes?**

  In both cases there were few enough left over to perform manual inspection

  Sample of output:

```
{ "address" : { "postcode" : "CF382HE" } }
{ "address" : { "postcode" : "Cf 31 1dq" } }
{ "address" : { "postcode" : "BS1 " } }
{ "address" : { "postcode" : "BS42eg" } }
......
{ "postal_code" : "NP" }
{ "postal_code" : "1" }
{ "postal_code" : "CF315FD" }
```

  The vast majority look like valid full or partial postcodes with a missing 'space'

	How many are there?	How many are valid UK postcodes?	What percentage are valid UK postcodes?	How many are invalid UK postcodes?	What percentage are invalid UK postcodes?	How many are valid 'partial postcodes'?	What percentage of the invalid postcodes are valid 'partial postcodes'?	Remaining invalid postcodes?
postal_code	7567	547	7.2	7020	92.8	6910	98.4	106
addr:postcode	5947	5848	98.3	103	1.7	66	64	26

- **How many contain lower case letters?**

  For consistency we will want to 'uppercase' postcodes, we can see its pretty rare

```
postal_code = 32
address.postcode = 0
```

Regional Accuracy

Do the observed postcodes belong in the geographic region under consideration? We can crudely determine this by consideration of the first two letters of the postcode.

There were few enough variations (12 for postal_code, 13 for addr:postcode) to consider manually. There was one 'nonsensical value', all the rest were valid for the region except:

NG – this is the postcode for Nottingham (well outside the region)

Inspection of the accompanying fields (e.g. town) suggested this should have been NP for Newport

General Tag Consistency Audit

With the ultimate aim of improving tag consistency I performed an audit of the observed tags against the OSM wiki features (i.e. essentially the best practice tag keys and values) [ref: 4]

Machine Readable Reference Data

The wiki feature set is available for download from Taginfo [ref: 5]

The format is a Sqlite3 database file [ref: 6] explains the process I underwent to extract usable reference data. Essentially the resulting format is a dictionary with a key for each 'tag key' with value containing the set of preferred 'tag values' (if any).

Audit Process

[ref: 7] is a python script that performs the audit. Roughly, for each tag key:

- Perform an exact match lookup (against the wiki reference data)
- If we **don't** find an exact match
    - Perform a 'sub-string match' (e.g. is our 'tag key' a sub-string of a OSM wiki tag key)

      Outputting each OSM wiki tag that is a substring match

      e.g key = street, preferred key = addr:street
    - Perform a closest match test

      Calculate 'Levenshtein edit distance' from each OSM wiki tag key and report the closest match (i.e. lowest number of characters to change in order to get an exact match)

      The main reason for doing this is to trap typos, maybe 1 or 2 characters out
- Alternatively, if we **do** find an exact match
    - Repeat the above process but this time for the 'tag value'

Audit Findings

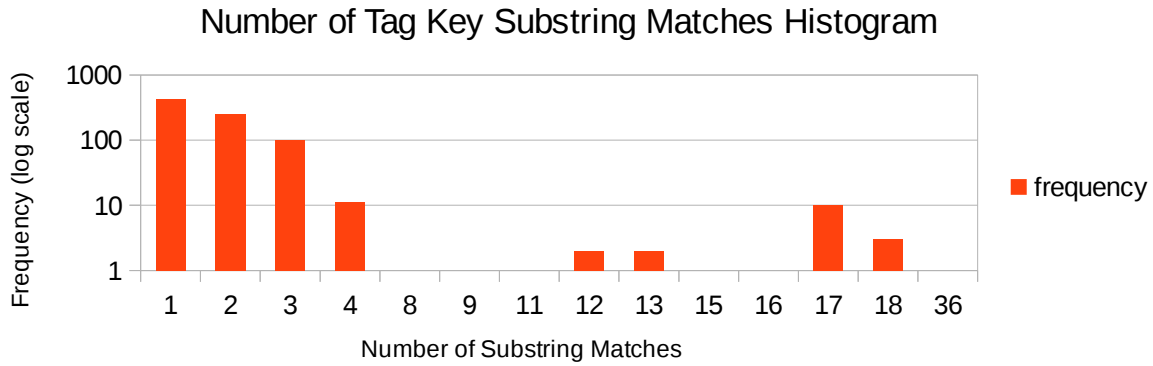
The audit produced a json file with a record containing the findings for each tag in the input set (see [ref: 8]). Additionally it produced a summary for the set as a whole which is described below.

Tag Key Findings

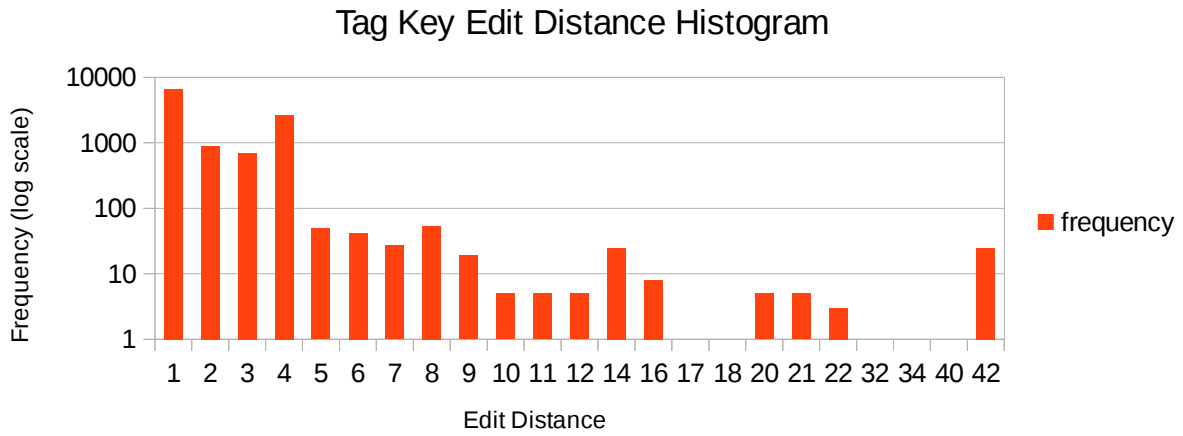
The table below shows the majority of 'tag keys' are in the preferred set but that there are still many cases; either sub-string matches or low edit distance that 'might' be candidates for change

	Count	Percentage	
Exact Match	831354.0	98.7	
No Exact Match	11206.0	1.3	
Substring Match	810.0	7.2	% of the 'No Exact Match' cases
No Substring Match	10396.0	92.8	% of the 'No Exact Match' cases
Levenshtein <=2	7575.0	67.6	% of the 'No Exact Match' cases

For completeness below is a histogram detailing how often we saw 1, 2 …. sub-string matches for a given 'tag key' (e.g. 254 'tag keys' in our data set matched 2 'tag keys' in the reference data). This illustrates how difficult it would be to then automatically choose a best practice alternative.



A further histogram shows the frequency of different minimum edit distances from the wiki set keys



Tag Value Findings

A similar treatment of 'tag values' (for cases when there was an exact key match) revealed the following (please note that in many cases the wiki set does not contain best practice 'tag values' – explicitly this was the case for 178,863 of our 'tag values, these are discounted from analysis)

	Count	Percentage	
Exact Match	451236.0	69.2	
No Exact Match	201255.0	30.8	
Substring Match	8469.0	4.2	% of the 'No Exact Match' cases
No Substring Match	192786.0	95.8	% of the 'No Exact Match' cases
Levenshtein <=2	182296.0	90.6	% of the 'No Exact Match' cases

Compared to the 'tag key' case, for 'tag values' we see a much lower exact match rate but perhaps surprisingly we see a high rate of 'low Levenshtein edit distance'

For brevity the equivalent histograms are omitted, they follow a similar pattern to the Key case.

Cleansing Strategy

Street Type

Apply the identified mappings to the identified valid street types

In total a mere 17 street values were changed!!!!!

Postcodes

- Map 'postal_code' onto 'addr:postcode' for consistency
- Map 'postcode' onto 'addr:postcode' for consistency
- Map 'source:postcode' onto 'source:addr:postcode'
- Map 'note:postcode' and 'note:postal_code' onto 'note:addr:postcode'

  (for the sake of our MongoDB document structure, include the above as fields in the address object)
- Introduce the space character into 'otherwise valid' full or partial postcodes
- Change NG postcodes to NP postcodes
- Upper case all postcodes

Tag Consistency

At this stage, though illuminating the audit is not refined enough to take remedial action against.
See 'Other Ideas' for brief discussion

Other Ideas

Following on from the 'Tag Consistency Audit' there may be merit in further developing this software into something that can provide quality recommendations to better align tags with the wiki set.

In particular the focus would be as a pre-load check for bulk data loads (much as we have been performing in this project).

It is unlikely that we could 'safely' automatically change values but we could rank and flag up the cases which appear to be obvious candidates for alignment.

As a first step we would have to improve the quality of both the sub-string audit and the edit distance measure.

For example, in the sub-string case we could weight a match on full sub-section of a key more highly (e.g. street would score highly against addr:street).

We could also enhance the Levenshtein edit distance measure by factoring in the length of the string rather than just the absolute measure (i.e. 2 character edit distance is more significant in a 20 character string than a 3 character string)

Conclusion

A modest improvement to a dataset that could already be considered 'quite clean' and some ideas that could lead to a more generic improvement in data consistency

Appendix A – Postcode Queries

Tag Key Count

```
mr = db.runCommand({
      "mapreduce" : "wwww",
      "map" : function() {
            for (var key in this) {
                  emit(key, 1);
            }
      },
      "reduce" : function(key, values) {
            return Array.sum(values);
      },
      "out": "wwww" + "_keys"
})

cursor = db.wwww_keys.find().sort({"value" : -1}).pretty()

while (cursor.hasNext()) {
  printjson(cursor.next());
}
```

Postcode Variant Observation

```
> db.wwww.find({"source:postcode" : {"$exists" : true}}).pretty()
```

Records that contain both major postcode variants

```
> db.wwww.count({"$and" : [{"address.postcode" : {"$exists" : true}},
{"postal_code" : {"$exists" : true}}]})
```

How many (of each variant of postcode) are there?

```
> db.wwww.count({"postal_code" : {"$exists" : true}})
```

How many are valid UK postcodes?

Postcode Regex (from uk gov publication https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/359448/4__Bulk_Data_Transfer_-_additional_validation_valid.pdf

```
^([Gg][Ii][Rr] 0[Aa]{2})|((([A-Za-z][0-]{1,2})|(([A-Za-z][A-Ha-hJ-Yj-y][0-9]
{1,2})|(([AZa-z][0-9][A-Za-z])|([A-Za-z][A-Ha-hJ-Yj-y][0-9]?[A-Za-z])))) [0-9]
[A-Za-z]{2})$
```

```
> db.wwww.count({"postal_code" : /^([Gg][Ii][Rr] 0[Aa]{2})|((([A-Za-z][0-]
{1,2})|(([A-Za-z][A-Ha-hJ-Yj-y][0-9]{1,2})|(([AZa-z][0-9][A-Za-z])|([A-Za-z][A-
Ha-hJ-Yj-y][0-9]?[A-Za-z])))) [0-9][A-Za-z]{2})$/})
```

How many are invalid UK postcodes?

```
> db.wwww.count({"$and" : [{"postal_code" : {"$exists" : true}},{"postal_code" :
```

```
{"$not" : /^([Gg][Ii][Rr] 0[Aa]{2})|((([A-Za-z][0-]{1,2})|(([A-Za-z][A-Ha-hJ-Yj-
y][0-9]{1,2})|(([AZa-z][0-9][A-Za-z])|([A-Za-z][A-Ha-hJ-Yj-y][0-9]?[A-Za-z]))))
[0-9][A-Za-z]{2})$/}}]})
```

How many are valid 'partial postcodes'?

```
> db.wwww.count({
      "$and" : [{
            "postal_code" : {
                  "$exists" : true
            }
      },
      {
            "postal_code" : /^([Gg][Ii][Rr]( 0)?)|((([A-Za-z][0-]{1,2})|(([A-Za-
z][A-Ha-hJ-Yj-y][0-9]{1,2})|(([AZa-z][0-9][A-Za-z])|([A-Za-z][A-Ha-hJ-Yj-y][0-
9]?[A-Za-z])))))( [0-9])?)$/
      }]
},
{
      "postal_code" : 1,
      "_id" : 0
})
```

What are the remaining invalid postcodes?

```
> db.wwww.findAll({"$and" : [{"postal_code" : {"$exists" : true}},{"postal_code"
: {"$not" : /^([Gg][Ii][Rr]( 0.*)?)|((([A-Za-z][0-]{1,2})|(([A-Za-z][A-Ha-hJ-Yj-
y][0-9]{1,2})|(([AZa-z][0-9][A-Za-z])|([A-Za-z][A-Ha-hJ-Yj-y][0-9]?[A-Za-z])))))(
[0-9].*)?)$/}}]}, {"postal_code" : 1, "_id" : 0})
```

How many contain lower case letters?

```
 >  db.wwww.count({"postal_code" : /[a-z]/})
```

Are the postcodes accurate for the given region?

Perform an aggregation on the first two letters

```
> db.wwww.aggregate([{"$group" : { "_id" : {"$substr": ["$postal_code",0,2]}}}])
```