Project 3: Open Street Map

Author: Denis Rudolf

Some words about the data. I downloaded an OSM XML file where a part of Western Germany is mapped (Düsseldorf area, latitude between 51.16696° and 51.23432°, longitude between 6.64838° and 6.87130°). The OSM file size is 230.532 MB. I would like to work with more data but for the auditing and cleaning step with xml.etree.cElementTree my RAM of 8 GB is not sufficient.

Step 1: Auditing and cleaning of the data

First, I checked street names for non-alphanumeric strings. I found some street names with non-alphanumeric values but most of them were correct (validated by Internet research). I corrected the incorrect ones on OSM. Then, I audited the postcodes and found them to be correct by cross-checking with Google maps. Then, I audited and cleaned the phone numbers putting them all in the same format (example: +4921311520). I also created a contact dictionary with the keys phone, fax and website. I also set all the values for the key = wheelchair to be lower case. Furthermore, I noticed that there is a key = fixme or key = FIXME (see program output, partly in German), but this has to be addressed for each case separately.

In [24]:

```
import xml.etree.cElementTree as ET
from collections import defaultdict
import re
import pprint


OSMFILE = "map_Duesseldorf_Neuss.osm"

problemchars_streetname = re.compile(r'[=\+\&<>;\"\?%#$@\,\t\r\n]', re.IGNORECASE)
problemchars_phone = re.compile(r'[=\/&<>;\'"\?%#$@\,\.\t\r\n]', re.IGNORECASE)
phone_re = re.compile(r"\+49|0049")
phone_re_0049 = re.compile(r"0049")
non_digits_re = re.compile(r"\D")
fixme_re = re.compile(r'fixme', re.IGNORECASE)
fixme_list =[]

def audit_street(street_name):
    m = problemchars_streetname.search(street_name)
    if m:
        print u"Problem with the street name: {}".format(street_name)
    return street_name

def audit_postcode(postcode):
    postcode = int(postcode)
    if postcode > 41564 or postcode < 40210:
        print u"Problem with the postcode: {}".format(postcode)
    return postcode

def audit_is_wheelchair(is_wheelchair):
    mapping = {"Yes":"yes", "No":"no", "Limited": "limited"}
    if set([is_wheelchair]) < set(["Yes", "No", "Limited"]):
        return mapping[is_wheelchair]
    else:
        return is_wheelchair

def audit_phone_number(phone):
    # remove hyphons
    phone = "".join(phone.split("-")[:])
    # remove white spaces
    phone = "".join(phone.split()[:])
    # remove slashes
    phone = "".join(phone.split(r"/")[:])
    # check if the country code is there
    if phone_re.search(phone):
        # take only the first phone number if there are more than one
        m = problemchars_phone.search(phone)
        if m:
            char = m.group()
            # print "Problem character: " + char
```

```
            # print "Problem character: " + char
            return phone.split(char)[0]
        elif phone_re_0049.search(phone):
            return "+49"+ phone.strip("0049")
        else:
            return phone
    elif non_digits_re.search(phone):
        return None
    else:
        return "+49" + phone.strip("0")

def audit_fixme(fixme):
    fixme_list.append(fixme)

def audit(osmfile):
    osm_file = open(osmfile, "r")
    for event, elem in ET.iterparse(osm_file, events=("start",)):

        if elem.tag == "node" or elem.tag == "way":
            for tag in elem.iter("tag"):
                if tag.attrib['k'] == "addr:street":
                    audit_street(tag.attrib['v'])
                if tag.attrib['k'] == "addr:postcode":
                    audit_postcode(tag.attrib['v'])
                if tag.attrib['k'] == "phone":
                    audit_phone_number(tag.attrib['v'])
                if fixme_re.search(tag.attrib['k']):
                    audit_fixme(tag.attrib['v'])
    osm_file.close()

if __name__ == '__main__':
    audit(OSMFILE)
```

In [53]:

```
print 'The number of key=fixme or key=FIXME is {}.'.format(len(fixme_list))
print 'Here are 10 examples: \n'
pprint.pprint(fixme_list[0:10])
```

```
The number of key=fixme or key=FIXME is 1672.
Here are 10 examples:

['Exact position - are the connections correct?',
 'name of exit',
 'exact position',
 'Warnton?',
 u'zu busrelationen hinzuf\xfcgen',
 'name of exit',
 'Am Bahnsteigdach befestigt',
 u'Welche Gaststt\xe4tten sind da aktuell?',
 'auch Bushaltestelle?',
 'opening_hours']
```

The following code is for data auditing, cleaning, inserting into the data model and to store the data in the json file. It is build upon the Udacity excercise in the case study.

In [ ]:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import xml.etree.cElementTree as ET
import pprint
import re
import codecs
import json


lower = re.compile(r'^([a-z]|_)*$')
lower_colon = re.compile(r'^([a-z]|_)*:([a-z]|_)*$')
problemchars = re.compile(r'[=\&<>;\'"\?%#$@\,\.\t\r\n]')

CREATED = [ "version", "changeset", "timestamp", "user", "uid"]


def shape_element(element):
```

```
    node = {}
    if element.tag == "node" or element.tag == "way" :
        # get the list of keys of the "node"/"way" dict
        keys = element.attrib.keys()
        # define the sub-dictionary
        dic_created = {}
        dic_address = {}
        dic_contact = {}
        dic_created.fromkeys(CREATED)
        _type = element.tag
        # check the existence of the keys
        if set(["id", "changeset","user","version","uid","timestamp"]) <= set(keys):
            # print element.attrib
            _id = element.attrib["id"]
            dic_created["changeset"] = element.attrib["changeset"]
            dic_created["user"] = element.attrib["user"]
            dic_created["version"] = element.attrib["version"]
            dic_created["uid"] = element.attrib["uid"]
            dic_created["timestamp"] = element.attrib["timestamp"]
            node = {"id":_id, "general_type": _type, "created": dic_created }
        if set(["lat", "lon"]) <= set(keys):
            lat = float(element.attrib["lat"])
            lon = float(element.attrib["lon"])
            node["pos"] = [lon,lat]
        # for ways: make a list of node refs
        if element.tag == "way":
            node_refs = []
            for tag in element.iter("nd"):
                # print "Key: {}, Value: {}".format(tag.attrib["k"],tag.attrib["v"])
                # print tag.attrib["ref"]
                node_refs.append(tag.attrib["ref"])
            node["node_refs"] = node_refs
        # iterate over the tags
        for tag in element.iter("tag"):
            # select tags with one colon
            if lower_colon.search(tag.attrib["k"]):
                colon_list = tag.attrib["k"].split(":")
                # create an address dict
                if colon_list[0] == "addr":
                    address_type = colon_list[1]
                    # audit and clean the street name if necessary
                    if address_type == "street":
                        address_value = audit_street(tag.attrib["v"])
                    # audit and clean the postcode if necessary
                    elif address_type == "postcode":
                        address_value = audit_postcode(tag.attrib["v"])
                    else:
                        address_value = tag.attrib["v"]
                    # ignore values with problematic characters
                    # if not is_problemchars(address_value):
                    dic_address[address_type] = address_value
                # create a contact dict
                elif colon_list[0] == "contact":
                    contact_type = colon_list[1]
                    # audit and clean the street name if necessary
                    if contact_type == "phone":
                        contact_value = audit_phone_number(tag.attrib["v"])
                    # audit and clean the postcode if necessary
                    elif contact_type == "fax":
                        contact_value = audit_phone_number(tag.attrib["v"])
                    elif contact_type == "website":
                        contact_value = tag.attrib["v"]
                    elif contact_type == "email":
                        contact_value = tag.attrib["v"]
                    else:
                        contact_value = tag.attrib["v"]
                    # ignore values with problematic characters
                    # if not is_problemchars(address_value):
                    dic_contact[contact_type] = contact_value
                # other cases with colon but without "addr"
                else:
                    value = tag.attrib["v"]
                    # ignore values with problematic characters
                    if not is_problemchars(value):
                        s = " "
                        key_string = s.join(colon_list)
```

```
                        node[key_string] = value
            # select tags with lower case
            if lower.search(tag.attrib["k"]):
                key = tag.attrib["k"]
                value = tag.attrib["v"]
                # ignore values with problematic characters
                if not is_problemchars(value):
                    if key == "phone":
                        # print key
                        node[key] = audit_phone_number(value)
                    elif key == "wheelchair":
                        node[key] = audit_is_wheelchair(value)
                    else:
                        node[key] = value
            # print problematic characters
            if is_problemchars(tag.attrib["v"]):
               # print tag.attrib["v"]
                pass
        # insert the address and contact dict into the node dict
        if dic_address:
            node["address"] = dic_address
        if dic_contact:
            node["contact"] = dic_contact

        return node
    else:
        return None

def is_problemchars(string):
    return bool(problemchars.search(string))

def process_map(file_in, pretty = False):
    # You do not need to change this file
    file_out = "{0}.json".format(file_in)
    data = []
    with codecs.open(file_out, "w") as fo:
        for _, element in ET.iterparse(file_in):
            el = shape_element(element)
            if el:
                data.append(el)
                if pretty:
                    fo.write(json.dumps(el, indent=2)+"\n")
                else:
                    fo.write(json.dumps(el) + "\n")
    return data

def test():
    # NOTE: if you are running this code on your computer, with a larger dataset,
    # call the process_map procedure with pretty=False. The pretty=True option adds
    # additional spaces to the output, making it significantly larger.
    data = process_map('map_Duesseldorf_Neuss.osm')
    print "Number of dictionaries: {}".format(len(data))
    pprint.pprint(data[0:9])

if __name__ == "__main__":
    test()
```

Step 2: Inserting the data into Mongo DB and querying the database

I inserted the json file 'map_Duesseldorf_Neuss.osm.json' in the database OSM as a collection map_Duesseldorf_Neuss. Let's have a look at the number of documents, nodes and ways first.

In [34]:

```
from pymongo import MongoClient
import pprint
import numpy as np


client = MongoClient('localhost:27017')
db = client.OSM
# some simple queries
size = db.map_Duesseldorf_Neuss.find().count()
print "The number of documents is {}.".format(size)
```

```
num_nodes = db.map_Duesseldorf_Neuss.find({"general_type": "node"}).count()
print "The number of nodes is {}.".format(num_nodes)
num_ways = db.map_Duesseldorf_Neuss.find({"general_type": "way"}).count()
print "The number of ways is {}.".format(num_ways)
```

```
The number of documents is 971624.
The number of nodes is 817709.
The number of ways is 153915.
```

Now, let's count the number if users.

In [8]:

```
pipeline = [{'$group': { '_id': '$created.user',}}
            ]
result_list = []
result = db.map_Duesseldorf_Neuss.aggregate(pipeline)
while result.alive == True:
    result_list.append(result.next())
print 'The number of unique users is {}.'.format(len(result_list))
```

```
The number of unique users is 1009.
```

Who are the most contributing users?

In [9]:

```
pipeline = [{'$group': { '_id': '$created.user',
                         'count': {'$sum': 1 }  }},

               {'$sort': {'count': -1}}]
result = db.map_Duesseldorf_Neuss.aggregate(pipeline)
[result.next() for i in range(0,10)]
```

Out[9]:

```
[{u'_id': u'black_bike', u'count': 262745},
 {u'_id': u'EinKonstanzer', u'count': 102627},
 {u'_id': u'rurseekatze', u'count': 93824},
 {u'_id': u'Antikalk', u'count': 92611},
 {u'_id': u'rabenkind', u'count': 91562},
 {u'_id': u'Sharlin', u'count': 55791},
 {u'_id': u'Athemis', u'count': 43618},
 {u'_id': u'j-e-d', u'count': 27520},
 {u'_id': u'mighty_eighty', u'count': 23296},
 {u'_id': u'Zyras', u'count': 21703}]
```

These were absolute numbers. I'm also interested in the percentage of contributed documents per user.

In [54]:

```
pipeline = [{'$group': { '_id': '$created.user',
                         'count': {'$sum': 1 }  }},
            {'$project': {'ratio': {'$divide':['$count', 1e-2*size]}}},

               {'$sort': {'ratio': -1}}]
result = db.map_Duesseldorf_Neuss.aggregate(pipeline)
result_list = [result.next() for i in range(0,10)]
pprint.pprint(result_list)

elem_sum = 0
for elem in result_list:
    elem_sum = elem_sum + elem['ratio']
print '\n Top 10 users contribute {0:.2f} % to the OSM for the Neuss/Duesseldorf area.'.format(elem_sum
)
```

```
[{u'_id': u'black_bike', u'ratio': 27.041839229990202},
 {u'_id': u'EinKonstanzer', u'ratio': 10.562419207430034},
 {u'_id': u'rurseekatze', u'ratio': 9.65641029863404},
 {u'_id': u'Antikalk', u'ratio': 9.531567766955119},
 {u'_id': u'rabenkind', u'ratio': 9.42360419256832},
 {u'_id': u'Sharlin', u'ratio': 5.742036013931315},
 {u'_id': u'Athemis', u'ratio': 4.489185116876487},
 {u'_id': u'j-e-d', u'ratio': 2.8323713699949775},
 {u'_id': u'mighty_eighty', u'ratio': 2.3976352992515624},
 {u'_id': u'Zyras', u'ratio': 2.233682988481141}]
```

```
Top 10 users contribute 83.91 % to the OSM for the Neuss/Duesseldorf area.
```

The conclusion here is similar to the sample MongoDB project that top 10 users contribute 83.91 % of the documents. Because I'm a selfish person I will also query for the number of my contributions (it could be more).

In [55]:

```
pipeline = [{'$group': { '_id': '$created.user',
                         'count': {'$sum': 1 }  }},
            {'$match': { '_id': 'DenisRudolf'}}
            ]
result = db.map_Duesseldorf_Neuss.aggregate(pipeline)
while result.alive == True:
    print result.next()
```

```
{u'count': 7, u'_id': u'DenisRudolf'}
```

Next, I'm constructing the geospatial index (2d sphere) in order to be able to make a query with the '$geoNear' operator. I'm interested in the number of key = fixme or key = FIXME being close to our house (lat 51.22085, lon 6.65236), i.e. within a radius of about 10 km (maxDistance = 1.5e-3 rad).

In [57]:

```
from pymongo import GEOSPHERE
from bson.son import SON
db.map_Duesseldorf_Neuss.create_index([("pos", GEOSPHERE)])
```

Out[57]:

```
u'pos_2dsphere'
```

In [58]:

```
# our house
coord = [6.65236,51.22085]
pipeline = [{'$geoNear':{ 'distanceField': 'pos','near': coord, 'spherical': True,
                          'query': { 'fixme': {'$exists': 1}}, 'maxDistance': 1.5e-3,
                          'distanceField': 'dist.calculated'}}
            ]
result_list = []
result = db.map_Duesseldorf_Neuss.aggregate(pipeline)
while result.alive == True:
    result_list.append(result.next())
print 'The number of dictionaries with key = fixme or key = FIXME is {}.'.format(len(result_list))
```

```
The number of dictionaries with key = fixme or key = FIXME is 100.
```

Now, I want to retrieve the minimum and maximum longitude and latitude coordinates and to calculate the area.

In [14]:

```
lat_max = 60.0
lat_min = 45.0
lon_max = 10.0
lon_min = 5.0

def get_pipeline(v_min, v_max, sort):

        min_datetime = '2016-01-01T00:00:00Z'
        version = '1'
        # gets values of pos between v_min and v_max with timestamp greater than min_datetime
        # and with version greater than 1
        # and with an address dict
        # sort = -1: descending, sort = 1: ascending
        pipeline = [{'$unwind': '$pos'},
                {'$match': {'$and':
                            [{'pos': {'$lt': v_max}}, {'pos': {'$gt': v_min}} ]}},
                {'$match': {'created.timestamp': {'$gt': min_datetime}}},
                {'$match': {'created.version': {'$gt': version}, 'address': {'$exists': 1} }},
                {'$project': {'pos':'$pos', 'address':'$address'}},
                {'$sort': {'pos': sort}}
                ]
        return pipeline
```

```
def get_min_max_coord(v_min, v_max):

    for sort in [-1,1]:
        result_list = []
        pipeline = get_pipeline(v_min, v_max,sort)
        result = db.map_Duesseldorf_Neuss.aggregate(pipeline, allowDiskUse = True )
        result_list = [result.next() for i in range(0,1)]
       # pprint.pprint(result_list)

        if sort == -1:
            print 'The max value {0:.5f}.'.format(result_list[0]['pos'])
        else:
            print 'The min value {0:.5f}.'.format(result_list[0]['pos'])

get_min_max_coord(lat_min, lat_max)
get_min_max_coord(lon_min, lon_max)
```

```
The max value 51.23432.
The min value 51.16696.
The max value 6.87130.
The min value 6.64838.
```

In [15]:

```
import numpy as np


def get_distance(lat_1, lat_2, lon_1, lon_2):
# all in km and radians
    coords = np.pi/180.0*np.array([lat_1, lat_2, lon_1, lon_2])
    R = 6371.0

    return 2.0*R*np.arcsin(np.sqrt( np.sin(0.5*(coords[1]-coords[0]))**2 + np.cos(coords[0])*np.cos(coo
rds[1])
                                    *np.sin(0.5*(coords[3]-coords[2]))**2 ))
# five decimal places = 1 m accuracy
d_south_north = get_distance(51.16696,51.23432,6.64838,6.64838)
d_west_east = get_distance(51.23432,51.23432,6.64838,6.87130)
area = d_south_north*d_west_east
print 'The south-north distance is {0:.3f} km.'.format(d_south_north)
print 'The west-east distance is {0:.3f} km.'.format(d_west_east)
print 'The total area is {0:.3f} km^2.'.format(area)
```

```
The south-north distance is 7.490 km.
The west-east distance is 15.520 km.
The total area is 116.249 km^2.
```

How many pharmacies are on the map?

In [59]:

```
pipeline = [{'$group': { '_id': '$amenity',
                         'count': {'$sum': 1 }  }},
             {'$match': {'_id': 'pharmacy'}},

                {'$sort': {'count': -1}}
           ]
result = db.map_Duesseldorf_Neuss.aggregate(pipeline)
result_list = []
while result.alive == True:
    result_list.append(result.next())

print 'The number of pharmacies on the map is {}.'.format(result_list[0]['count'])
print 'The average number of pharmacies per km^2 is {}.'.format(result_list[0]['count']/area)
```

```
The number of pharmacies on the map is 124.
The average number of pharmacies per km^2 is 1.06667351055.
```

How many amenties have an access for a wheelchair and which ones?

In [46]:

```
pipeline = [{'$group': { '_id': '$wheelchair',
                         'count': {'$sum': 1 }  }},
                {'$sort': {'count': -1}}
```

```
            ]
result = db.map_Duesseldorf_Neuss.aggregate(pipeline)
result_list = []
while result.alive == True:
    result_list.append(result.next())

pprint.pprint(result_list[:])
elem_sum = 0.0
for elem in result_list[1:]:
    elem_sum = elem_sum + elem['count']
print '\n'
print '{0: .2f} % of the amenities with a wheelchair value have a wheelchair access.'.format(100.0*resu
lt_list[1]['count']/elem_sum)
print '{0: .2f} % of the amenities with a wheelchair value don\'t have a wheelchair access.'.format(100
.0*result_list[2]['count']/elem_sum)
print 'For {0: .2f} % of the amenities with a wheelchair value the wheelchair access is limited.'.forma
t(100.0*result_list[3]['count']/elem_sum)
print 'For {0: .2f} % of the amenities with a wheelchair value the wheelchair access is unknown.'.forma
t(100.0*result_list[4]['count']/elem_sum)
```

```
[{u'_id': None, u'count': 968150},
 {u'_id': u'yes', u'count': 1673},
 {u'_id': u'no', u'count': 977},
 {u'_id': u'limited', u'count': 805},
 {u'_id': u'unknown', u'count': 19}]


 48.16 % of the amenities with a wheelchair value have a wheelchair access.
 28.12 % of the amenities with a wheelchair value don't have a wheelchair access.
For  23.17 % of the amenities with a wheelchair value the wheelchair access is limited.
For  0.55 % of the amenities with a wheelchair value the wheelchair access is unknown.
```

The city should definitely do more for the the wheelchair users.

In [51]:

```
pipeline = [{'$match': { 'wheelchair': 'yes', 'amenity':{'$exists':1}}},
            {'$group': { '_id': '$amenity',
                         'count': {'$sum': 1 }}},
            {'$sort': {'count': -1}}
           ]
result = db.map_Duesseldorf_Neuss.aggregate(pipeline)
result_list = []
while result.alive == True:
    result_list.append(result.next())

print 'Tope ten of amenities with a wheelchair access.'
pprint.pprint(result_list[0:10])
```

```
Tope ten of amenities with a wheelchair access.
[{u'_id': u'parking', u'count': 98},
 {u'_id': u'pharmacy', u'count': 49},
 {u'_id': u'restaurant', u'count': 48},
 {u'_id': u'bank', u'count': 47},
 {u'_id': u'fast_food', u'count': 46},
 {u'_id': u'toilets', u'count': 42},
 {u'_id': u'cafe', u'count': 25},
 {u'_id': u'pub', u'count': 20},
 {u'_id': u'fuel', u'count': 14},
 {u'_id': u'post_office', u'count': 14}]
```