Elections and OpenData : on the tech side...

Submitted on 22 Apr 2014 by Mathieu Gaborit

This blog post has been written following the creation of a visualization map of the last municipal elections in Le Mans. The map was created based on an idea from @fblain and myself (@matael) with help from the HAUM and @jblb_72. (The raw data used can be obtained from http://extra.lemans.fr/elections).

This is not the first time I have worked on Le Mans' data, having previously worked on SETRAM's data (Le Mans transportation service), but this time around I turned my attention to the latest poll data.

Before diving into the data, I should let you know that all the scripts used in the project are available on Github.

Poll Data

In order to analyze the latest poll results, we need to find raw data that relates to it. More precisely, to create a vizualisation, we need data with the smallest granularity.

In Le Mans, we were able to find (thanks to feedoo) a website dedicated to election data, with data ordered by polling division.

If you open the source code of the page sent back by the server for the `first round`, you would be disappointed. However, we'll take a look at the only clue we have, a javascript file named 'soluvote.js'.

The script is a little long, but elements of interest can be found on lines 15, 205 and 236:

```
// line 15
    $.ajax({
        url: "xml/Election.xml?d=" + (new Date()).getTime(),
        // ...
    // line 205
    fichier = $(this).find('fichier').first().text() + ".xml";
    // line 236
    $.ajax({
        url: "xml/" + fichier + "?d=" + (new Date()).getTime(),
        // ...
```

These lines invite us to add content at the end of the URL:

- When we add '/xml/' - sent back a 403 forbidden error.
- When we add '/xml/Election.xml' (without adding any date), we get a XML file that appears to be a lookup table binding the vote offices (sectors) with their associated results files.
- Finally, when we add '/xml/' followed by one of the file names found in 'Election.xml', we get access to detailed results 'per-office'.

Let's try to automate this process ...

General Configuration

The 'config.py' file contains the name of the electoral round that interests us (here, it will be *premier_tour* or *second_tour*). I've also added to this file a list of engaged candidates and associated colors for all of them.

Here's what 'config.py' looks like:

```
    tour = 'second_tour'
    listes = {
        'premier_tour': {
            1: "LISTE D'UNION DE LA DROITE ET DU CENTRE (Alain PIGEAU)",
            2: "LUTTE OUVRIERE FAIRE ENTENDRE LE CAMP DES TRAVAILLEURS (Yves CHEÈRE)",
            3: "LE MANS RENOUVEAU CITOYEN (Ariane HENRY)",
            4: "AVEC VOUS POUR LE MANS (Christelle MORANÇAIS)",
            5: "LE MANS POUR TOUS (Jean-Claude BOULARD)",
            6: "CARTON ROUGE (Pascal LE PORT)",
            7: "LE MANS BLEU MARINE (Louis NOGUÈS)",
            8: "ALTERNATIVE PROGRESSISTE SOLIDAIRE (Michel PEZERIL)"},
        'second_tour': {
            1: "AVEC VOUS POUR LE MANS (Christelle MORANÇAIS)",
            2: "LE MANS POUR TOUS (Jean-Claude BOULARD)",
            3: "LE MANS BLEU MARINE (Louis NOGUÈS)"}
    }
    colors = {'premier_tour': {
            1: "#2E9AFE",
            2: "#FF0040",
            3: "#B40404",
            4: "#0101DF",
            5: "#FE2E9A",
            6: "#FF0000",
            7: "#08088A",
            8: "#B40431"
        },
        'second_tour' : {
            1: "#0101DF",
            2: "#FE2E9A",
            3: "#08088A",
        }
    }
```

Scrapping

This time, the scrapping part wasn't too hard to implement. Data was formatted in order to be processed by a (javascript) script, and it made a big difference.

Here is the script ('scrap_scrutin.py', comments should be enough to understand it):

```
    import json
    from bs4 import BeautifulSoup
    from requests import get
    from config import tour
    base_url = 'http://extra.lemans.fr/elections/{}/xml/'.format(tour)
    # get a list of offices
    get_liste = get('{}{}'.format(base_url,'Election.xml'))
    bureaux_soup  = BeautifulSoup(get_liste.text).find_all('bureau')
    bureaux = {}
    for b in bureaux_soup:
        # isolate the number of the current office
        num = int(b.find('numero').text)
        print("Scraping office {}...".format(num))
        # get the XML file of office-related results
        fichier = "{}.xml".format(b.find('fichier').text)
        # extraction of resultts
        candidats = BeautifulSoup(get("{}{}".format(base_url,fichier)).text).find_all('candidat')
        c_results = {}
        for c in candidats:
            c_results[int(c.find('intituler').text.split('-')[0])] = float(c.find('pourcentage').text.split('%')[0].replace(',','.'))
        # append it all to a general list
        bureaux[num] = c_results

    # save
    savefile = "data/{}.json".format(tour)
    print('\nSaving to : {}'.format(savefile))
    with open(savefile, 'w') as f:
        f.write(json.dumps(bureaux))
    print("Everything's OK... Quit.")
```

The beginning and end of the script are the retrieval of the list of offices, and the saving of the final cleaned results to a file (caching to use it later).

The for-loop iterates over the list of offices and for each of them, look up for the 'num' field first (to bind offices and sectors), and for 'fichier' (file in French). Once we have the name of the results file, we just use a GET request and BeautifulSoup to extract interesting data of those results. Finally, we use a hashmap between candidates and their result on the poll.

Where are those offices?

Le Mans is not known to encourage a wonderful Open Data policy, although I hope this will one day change. Actually (and I already wrote about it here in French), data is available hidden inside a zip, after a 'manual' license validation and under URLs that discourage any time of automation.

So we can access the zip file at this URL: http://www.lemans.fr/page.do?t=2&uuid=10A48915-550EA533-1F82E3AA-D697BAF8 .

Once uncompressed, we accessed the file 'csv/BUREAUX_COTE.csv' and we extracted the columns 'COMMUNE', 'ADRESSE' and 'SECTEURS'. Then we split the last of them to get the sector list associated to each office and we extrapolated the postal code from the first one.

We have the following script:

```
import csv
import json
# Extraction of data out of LeMans "OpenData" files
#
# We try to generate a liste of addresses in order to batch-geocode them
filename = "data/BUREAUX_VOTE.csv"
with open(filename) as f:
    reader = csv.reader(f, delimiter=";")
    reader.next()
    crossref_bureaux_oldaddr = {}
    error_count = 0
    for l in reader:
        if l[3] == 'Le Mans': # extrapol. du CP
            cp = 72000
        else: error_count += 1
        addr_str = "{}, {} {}, France".format(l[6],cp,l[3]) # full address formatting
        # change the SECTEURS field list into a real list
        crossref_bureaux_oldaddr[addr_str.decode('latin-1')] = [int(_) for _ in l[10].split(':')[1].split(',')]
        print(addr_str)

print("\n\nErrors : {}".format(error_count))

# save crossrefs
savefile = 'data/crossref.json'
print('Saving crossref file to {}...'.format(savefile))
with open(savefile, 'w') as f:
    f.write(json.dumps(crossref_bureaux_oldaddr))
```

We exported the cross-references dictionary to re-use it later.

Geocoding

In order to place those points on a map, we had to convert postal addresses to GPS coordinates - and decided to use this site to do so (and many thanks to them, we probably made their API limit explode).

Note: The process consisting in the conversion of postal addresses to GPS coordinates is called 'geocoding'.

Using this site, we received a CSV formatted this way:

```
lat;lon;used address; submitted address
```

From this file, we exported the coordinates, which are then linked to the corresponding sector through the cross-references table:

```
import sys
import csv
import json
from config import tour
# Creation of a clean GeoJson file
#
# This script follows extract_OD.py, it uses the data exported by the
# geocoder and change them into a list of features ready for an import
# on a map.
# file exported by the geocoder
filename = "data/bureaux_vote_coords.csv"
# loading of the crossref file
print('Loading crossref from data/crossref.json')
try:
    with open('data/crossref.json') as f:
        crossref_bureaux_oldaddr = json.load(f)
except IOError:
    print('data/crossref.json not found...\nPlease run extract_OD.py before')
    sys.exit(0)
# reading of CSV and creation of the GeoJSON
geolist = []
with open(filename) as f:
    reader = csv.reader(f,delimiter=";")
    for l in reader:
        geolist.append(
            {
                "type": "Feature",
                "geometry": {
                    "type": "Point",
                    "coordinates": [l[1], l[0]]
                },
                "properties": {
                    "name": "Bureau de vote",
                    "secteurs": ','.join(map(str,crossref_bureaux_oldaddr[l[3].decode('utf-8')]))
                }
            }
        )

# saving
savefile = "data/bureaux_vote_coords.json"
print('Saving GeoJSON list to {}...'.format(savefile))
with open(savefile,'w') as f:
    f.write(json.dumps(geolist))
```

Link office-results

Now that we had both the list of poll offices and results by sector, we could start to mix these up:

```
import json
from config import listes, colors, tour
# poll data
print('Loading poll data...')
with open('data/{}.json'.format(tour)) as f:
    bureaux = json.load(f)
    bureaux = {int(k):v for k,v in bureaux.items()}
# offices coordinates (GeoJSON fomat)
print('Loading coords of offices')
geo_filename = "data/bureaux_vote_coords.json"
with open(geo_filename) as f:
    geolist = json.load(f)
# adding results
```

```
for i in range(len(geolist)):
    # iteration over offices
    data_files = geolist[i]['properties']['secteurs']
    # create a dict with the correct number of lists and 0% to each
    results = {_+1:0 for _ in range(len(listes[tour]))}
    # fetch the sectors associated to the current office
    secteurs = map(int, data_files.split(','))
    # grouping of several sector on one office
    for num in secteurs:
        for k,v in bureaux[num].items(): results[int(k)] += v
    # normalisation by the number of grouped sectors
    for k,v in results.items():
        geolist[i]['properties'][listes[tour][k]] = "{} %".format(v/len(secteurs))

# save
savefile = "data/bureaux_vote_results_{}.json".format(tour)
print('Saving to {}...'.format(savefile))
with open(savefile,'w') as f:
    f.write(json.dumps(geolist))
```

When the resulting file is imported into OSM, it draws the round results as shown here.

Sectors

A more significant (and visual) aspect comes with the visualisation of sectors, colored with different criteria. In our case, the drawing of the sectors themselves was a difficult thing... let's see why.

Raw data

Raw data (as given by the city administration) is a single 'pack' called `Voies par secteurs` (streets by division).

In this pack, you can find several files corresponding to different types of division, a particularly useful one being: 'VOIES_PAR_BUREAU_VOTE.csv' (street by poll offices).

The CSV file is using this header:

```
RIVOLI;VOIE;MINIMUM_PAIR;MAXIMUM_PAIR;MINIMUM_IMPAIR;MAXIMUM_IMPAIR;NUM_BV;BUREAU_DE_VOTE
```

So we proceeded this way:

- For each sector and each street of the sector, we isolated exact addresses of the first and last number of the street
- We wrote a CSV file using the geocoding website (by copy/paste)
- We changed the returned data into mapable data (using 'csv2geojson.py')
- We traced a polygon around the markers by hand
- We checked our polygon using an 'official' map that someone gave to us (but which isn't accurate enough to be processed by a script directly)
- We then returned to step 1 for the next division...

This is the reason feedoo and I have lost several nights plotting polygons on a map and ruining API limits of a poor geocoding website. (Some will say that we could have written a script using an API to automate all of this. It's true, but not necessarily easier)

Map Plotting and export

Once we finished the plotting of all the 97-poll divisions of Le Mans, we re-exported the polygons under a GeoJSON format.

Let's now see how we exploited this file to include results in it.

Color time!

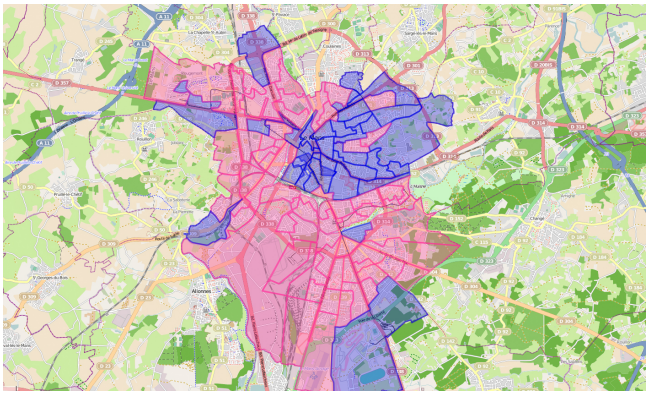
We decided that visualizing the map of results with colored sectors would be more comprehensible. Note that the script is closely similar to the 'per office' one:

```
import json
from config import listes, colors, tour
# first, we read the list of sector and turn it into a usable data structure
sect_filename = "data/secteurs.json"
with open(sect_filename) as f:
    tmp = json.load(f)
    # we want a hashmap : sector num -> list of coords of the polygon
    map_secteurs = {int(_['properties']['name'].split(' ')[1]):_['geometry']['coordinates'] for _  in tmp['features']}
# loading of poll data
print('Loading poll data...')
with open('data/{}.json'.format(tour)) as f:
    bureaux = json.load(f)
    bureaux = {int(k):v for k,v in bureaux.items()}

# create an empty GeoJSON features list
geolist = {"type": "FeatureCollection", "features": []}
for s in map_secteurs.keys():
    # for each sector, create a polygon feature with the right
    # coordinates and the name of the sector
    new_feature = {
        'type': 'Feature',
        'geometry': {
            'type': 'Polygon',
            'coordinates': map_secteurs[s]
        },
        'properties': {
            '_storage_options': {},
            'name': 'Secteur {}'.format(s)
        }
    }
    # extract results for the sector from all the data
    res = bureaux[s]
    res = {int(k):int(v) for k,v in res.items()}
    # colors of sector
    # sort candidates by votes percentage
    res_sortedkeys = sorted(res, key=res.get, reverse=True)
    # take the first element of the sorted list and make the sector
    # correctly colored depending on who (s)he is
    color = colors[tour][res_sortedkeys[0]]
    new_feature['properties']['_storage_options']['color'] = color
    # score of each of them (to be shown in a pop-up)
    for k,v in res.items():
        new_feature['properties'][listes[tour][k]] = "{} %".format(v)
    # append to list
    geolist['features'].append(new_feature)

# save
savefile = "data/secteurs_results_{}.json".format(tour)
print('Saving to {}...'.format(savefile))
with open(savefile,'w') as f:
    f.write(json.dumps(geolist))
```

The result is also shown on the layers of this map.



What's next?

There's a lot more to do! First of all (as I think you will now have gathered) access to data is not an easy thing. Moreover, the "OpenData" of Le Mans is not up-to-date, nor complete - actually, without the "official" map given to us, the vast majority of sectors would have been drawn incorrectly.

Having said that, we want this poll visualization to take all its meaning – and we MUST contextualize it! We would need to analyse data of previous polls, cross all the data we already have with demographic of socially related data, cultural aspects etc, and in short get them back in the social context that they come from.

On this side, there's still a lot of work to do...