Detecting Geographical Flows with Twitter in Switzerland

Loïs Talagrand & Paul Nicolet

***Abstract*—With the expansion of mobile devices and applications, it is easy nowadays to collect huge amounts of geographical data generated by users representing a large portion of the population. This data is processed by data scientists in order to find results which couldn't be found earlier with classical research. In this direction, this project aims to use geographical data from Twitter in order to find insights about the geographical movements in Switzerland.**

I. Introduction

We can identify two main final goals for this project: a *scientific* goal and a *learning* one.

The scientific goal consists in building a full pipeline in order to perform an analysis on a Twitter dataset, from raw data to visualization, about geographical flows in Switzerland. The motivation is quite straightforward. Indeed, Twitter, the famous platform allowing people to post and share short messages, often collects geographical data about its users. The popularity of the social network is the key to gather locations of an important sample of the population. Thus, performing an analysis on this kind of dataset might give good ideas about how people move in Switzerland. Of course, even if the size of the dataset seems big, we have to keep in mind that this is probably far from representing the majority of the Swiss population.

The learning goal consists in gaining experience and confidence in the data analysis field. Going from raw data to final visualization is a complex and long path, which we haven't had the opportunity to completely follow yet with a large scale project. In addition to the *Python* analysis tools, such as *pandas*, which we used during the semester, we wanted to gain skills in the tools in which we don't have experience at all like *Spark* or *d3.js*. It was then very important for us to think of a project which would need heavy calculations and visualization.

This project takes place in the context of the Applied Data Analysis course of EPFL, taught be Prof. Catasta.

II. Models and Methods

We found that completing this project requires to go through three main challenges: designing a model for flow detection, learning cluster computing, and developing an interesting visualization with an unknown library. We describe them in the following sections.

A. Detection Model

Going from tweets to geographical flows is not a trivial task. You can find the development process for each main feature of the model under the *notebook* folder of the repository.

In order to define the building blocks of our algorithm, we dedicated some time for the original design of the model in order to make sure to start with solid enough bases. You can find the original reflection following this link. In short, we thought that flows should exist between important cities, and that small moves between less important locations should be ignored.

The next step was to be able to properly locate tweets. Even if each tweet is located using a pair of coordinates, we needed a way to generalize those and manipulate locations using more general pieces of information. We introduced *nodes*, representing important cities. Each tweet would be associated with a global node. Two questions arise: how can we build these nodes and how can we associate tweets and nodes ? To build nodes, we used open source data from Geonames, an open database for geographical data. The process consists in extracting the most populated cities of Switzerland and building nodes out of them. We also considered bordering countries, as they might play an important role in Swiss flows, by taking the closest foreign cities to previously calculated Swiss nodes, above a given population threshold. Associating tweets to nodes was a pretty easy task once the nodes are defined. We simply link a tweet to its closest node, if the tweet is in the city's radius (see this notebook).

The main part of the detection algorithm is to actually build the flows. It turned out that it's quite difficult to build with confidence a perfect and reliable algorithm for this task, as it relies on a lot of assumptions. The global idea is the following: isolate tweets of each user, build pairs of tweets to analyse and decide for each pair, depending on several conditions, if it can potentially represent a flow or not (see this notebook). Redundant flows across users are selected as main flows and compose the final results. The main issue with this task is that it requires a lot of parameters to select, and no true value to guess for each of them given we are not expert in this field: number of nodes, number of foreign nodes, duration interval between two tweets... Playing with these values give different results, and we were unable to decide which are the bests. This is why we decided to follow another strategy. Given that the goal is to look for geographical patterns in Switzerland, it can be interesting to provide users with the key to play with the parameters which define the detection of these patterns, and let them decide on their own what are the determinant criteria for this task.

It seems like a good contract between us and the users, and allows us to play with two new tools. Indeed, generating a lot of results with different parameters would require a lot of computing power, and furthermore, visualize parametrized data would imply some form of dynamic visualization.

B. Cluster Computing

Being able to change parameters means running the algorithm with multiple parameter sets. This requires of course a lot of computing power. It takes around 5 seconds to run the algorithm with one single set of parameters on the sample data on a simple laptop. So we definitely needed more computing power to be able to run it potentially hundred of times on the whole dataset, which is approximately 5 gigabytes. Fortunately, the course staff provides a cluster with large scale computing tools like *Spark* and *HDFS*.

Given that none of us had prior experience with large scale computing, it has been a real challenge to familiarize ourselves with this new environment and adapt the algorithm to support *Spark* parallel computing model. The main algorithm was first developed in raw *Python*, using classic data structures like dictionaries, and designed to be run with simple sequential loops. Thus we had to learn bases of *Spark* programming to be able to translate our design.

One part of the work consisted in adapting the data structures used throughout the algorithm to make them compatible with *Spark*. Indeed it turns out that *Spark* doesn't really like dictionaries. We basically made the necessary to get a final structure using key/value stores. The second part was to adapt the algorithm itself in order to make our functions compatibles with the *Spark map* and *reduceByKey* operations. All of this work was of course designed to process the cleaned tweets, distributed across the nodes by *HDFS*. The computed results are then written back to *HDFS* in a *pseudo-json* format, which needs a little bit of processing before being consumed by the visualization.

C. Visualization

It turned out that visualizing flows is easier said than done. Indeed it is pretty straightforward to imagine arrows and nodes on the screen in order to see what happens, but there is no easy way to implement this since there is no library built for this especially. So we decided to take a look at *d3.js*, the well-known *JavaScript* library for visualizing data in a browser. The tricky part was to quickly learn the basics of this library in order to implement the components that we needed from scratch, this consists in assembling basic shapes to create more complex ones. Once the basic arrows and nodes are coded, some code is necessary in order to translate flows and nodes from the *json* format to the visual result. The dynamic part of the visualization consists in selecting which flow file to load. Inputs are available to the user, in order to choose the parameters of the flows to display. These parameters are the ones discussed above in the model description, we generated a few hundred of files with the *Spark* job in order to provide a sample visualisation. Of course most of the possible optimization concerns the general design of the visualization, which can be quite tricky when you are not used to web design.

The visualization of the sample results is available on this page. Note that the date parameters are not relevant in this sample visualization since files are generated from tweets very close in time.

III. Results

From a scientific point of view, this project did not bring what we expected. Indeed we ran into a lot of troubles deploying the final *Spark* job to the cluster near the end of the project. As a result, we couldn't make it in time to process the entire dataset and collect flows computed from all the available tweets. *Spark* is apparently not very friendly with randomized hashing of *Python3*, and we could not fix the problem with the course staff before the deadline. We decided to focus on finishing the visualization by generating flows with the sample data, as a proof of concept, showing that we can potentially obtain results out of this project.

IV. Summary

This is the first time we had the opportunity to develop a data analysis project from raw data to visualization, and we can say that we reached this goal. Indeed, we encountered quite a few problems during the development, the biggest one being the deployment on the cluster, but we managed to turn in a working final product.

Even if the *Spark* job couldn't be run on the cluster, this script has been developed until success since it produced the results from the sample data on a local *Spark* distribution.

Developing the visualization using *d3.js* has shown that creating from scratch its own visualization for a special project is a huge amount of work and requires a lot of time.

To conclude, we can say that even though the final results are not as shiny as expected, we are quite satisfied of the accomplishments, since it brought us a lot from a learning prospective.

Acknowledgements

We would like to thank the Applied Data Analysis course staff, and in particular Michele Catasta, for their work during the semester, as well as for the hardware resources.