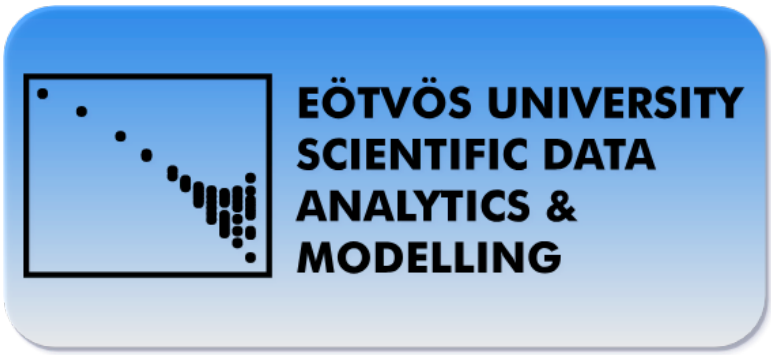
ELTE Faculty of Science



Twitter localization

Alex Olar

2018

Abstract

During the years the university has built a database using Twitter's public API and collected more than 3 TB of data to analyze. I got access to this dataset and attempted to predict the location of users based on their followers' location.

The problem itself is not a machine learning problem but an analytical approach and making assumptions about the behavior of users. Therefore I put an emphasis on visualization since the most crucial thing one does before any algorithmic solution is to analyze the visualized dataset.

Contents

I. The dataset

I was provided with an undirected, full graph with bidirectional data of followers. Meaning that the dataset I was given included all the users and their followers who they followed as well.

Assumption I.: This is already an assumption that the users themselves follow many celebrities, stars, artists whom are not following them, therefore their location is not relevant.

The graph was stored in SQL with and the following properties were present: *userId*, *followerId*, *userLon*, *userLat*, *followerLon*, *followerLan*, *dist*.

I have not used the distance metric at all because I did not need in in my process of finding the best location to predict. However, this kind of dumped data layout was not very useful for my data exploration since I didn't exactly have the followers' list of a user and therefore I could not say anything about their location.

Firstly I built a table with acquiring all the *userId*s and their geographical locations. This is called later on *Twitter_ User*. I acquired this by running the following query:

Listing 1: Building the user table

```
INSERT INTO Twitter_User (ID, lon, lat)
    SELECT DISTINCT id as ID, lon, lat
    FROM follower_graph;
```

On this table I set up *ID* as *PRIMARY KEY* since the indexing of the data, which was not present on the follower graph table made the querying process way faster. I copied that table as well and set the *(userId, followerId)* pair as *PRIMARY KEY* and did a join on the data tables in order to acquire all the data from the DB in order. This process resulted in $\approx$ **94 million)** rows which is approximately 5 GBs of data. The *Twitter_ User* table contained $\approx$ **3.2 million** users.

Listing 2: Building the user table

```
SELECT follower.fid, follower.flon,
  follower.flat, follower.dist,
  twitter_user.lon, twitter_user.lat,
  twitter_user.ID
FROM alex..Twitter_User as twitter_user
INNER JOIN alex..follower_graph as follower
  ON twitter_user.ID = follower.id
ORDER BY twitter_user.ID
```

After acquiring the data it was obvious how to proceed. I created a dictionary where the keys were the user ids and one record contained all the follower ids and locations of the followers and as many types the user's location as well, this part was sadly not optimized.

Having gathered all the necessary information about the users I could move on the the general algorithm of localization.

II. The essence of localization

Using the locations of the user's followers I wanted to select the largest group of closest followers and get the mean of their locations. The problem and the solution is well described but the implementation is not that straightforward since the locations are provided as geographical coordinates, so not only is clustering not trivial on the surface of the globe but also it is somewhat complicated to get the mean location of the users.

Since the number of clusters could not be estimated beforehand and I wanted to select the most densely populated neighbor cluster I used **DBSCAN** [5] clustering which is density based and can relatively well cluster this type of data. It was also a huge advantage that the *sklearn* [4] package included the *Haversine* metric.

The *Haversince formula* determines the great-circle distance between two points on a sphere given their longitudes and latitudes [6]. The haversine function of an angle ϑ is given in the form:

$$hav(\vartheta) = sin^2\Big(\frac{\vartheta}{2}\Big) = \frac{1 - cos(\vartheta)}{2}$$

Where ϑ is the angle - longitude or latitude - of a point. The central angle Θ is defined as the fraction of the distance of the points over the radius **R** of the sphere which is *6371 km* this case. The Haversine formula is described then as:

$$hav(\Theta) = hav(\varphi_2 - \varphi_1) + cos(\varphi_1)cos(\varphi_2)hav(\lambda_2 - \lambda_1)$$

The distance can be derived from this formula. *DBSCAN* takes a few more arguments such as minimal distance to cluster on which was set to $\frac{5.}{6371.}$ ($\equiv$ 5 km) and the minimum samples in a cluster was set to 5. The longitudes and latitudes were converted to radians and the clustering was done.

The algorithm generates outliers as well and sometimes there are no clusters at all. In all those cases I just converted the geographical coordinates to cartesian coordinates, averaged them and converted back to geolocations. For proper conversion I used the *astropy* package which provided an implementation of conversion.

On the other hand, when the clustering was done properly I selected the largest cluster of the followers of a user and averaged their locations as the above described manner.

Finally I created a dataset with the predicted and actual locations of the users and generated 50 filtered dataset based on maximum distance between the predicted and actual locations. With this method I acquired some kind of accuracy metric:

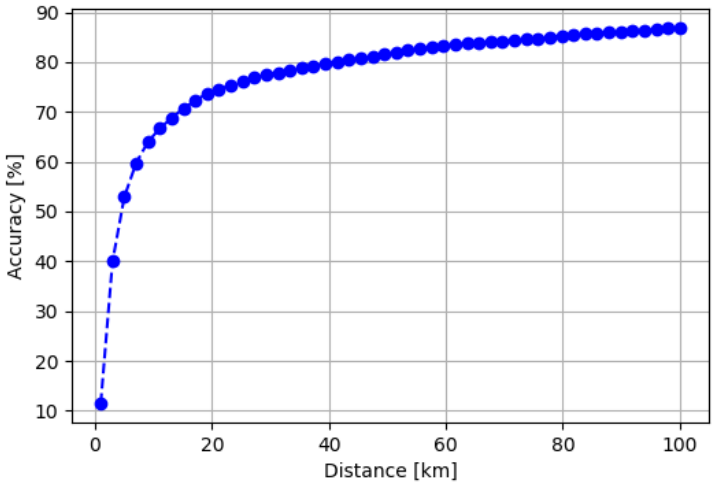


Figure 1: Accuracy on maximum distance allowed

The accuracy of prediction is really good for 100 km, although, it is not that bad for 1 km either since it acquired more than 10 % precision which I assume could be much higher making the prediction only in a city f.g. *New York*. Lowering the scale from 0 to 1 km:

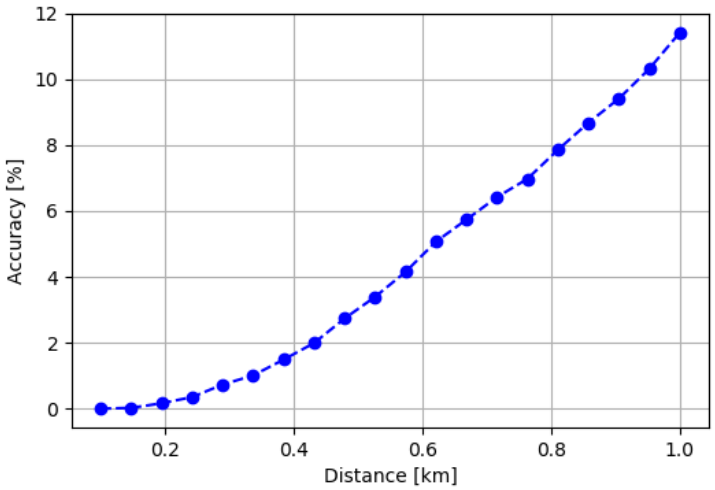


Figure 2: Accuracy on maximum distance allowed - finer scale

There is no point in raising the maximum distance more therefore I am not calculating the accuracy in that case. The accuracy was simply acquired by taking the maximum number of predicted points and dividing by it on the filtered points. Unfortunately the distance calculation in this period was not based on the Haversine metric but on basic cartesian distance, although, it is not completely inaccurate in this 0 - 100 km scale.

III. The visualization

The visualization tool is reachable at `https://twitter-localization.herokuapp.com`. It consists of a data providing backend, hosted on Heroku as well. The backend has saved *csv* files on different maximum distances and each of those is fed to the Angular frontend using Leaflet.js [1] as visualization tool. This package provides zoomable, and scaleable maps in JavaScript for the web. Little Twitter icons show the predicted and actual locations of users (red and blue icons) and they are connected by a straight blue line.

The tool provides a slider with which it is possible to change the maximum distance allowed and shows the accuracy metric and displayed number of points on the map. I include screen shots but it can be tested on the fly during the lecture.

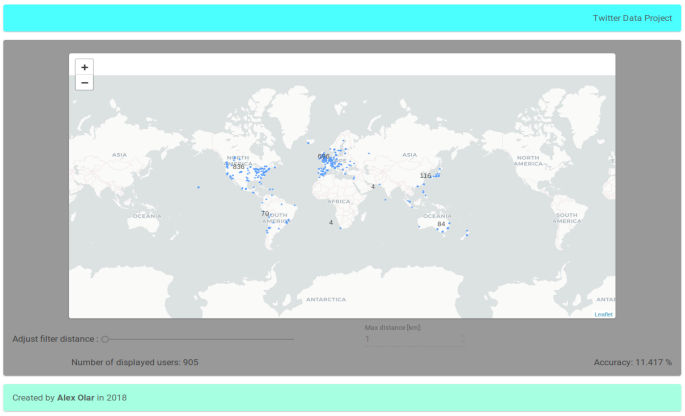


Figure 3: Default value - full

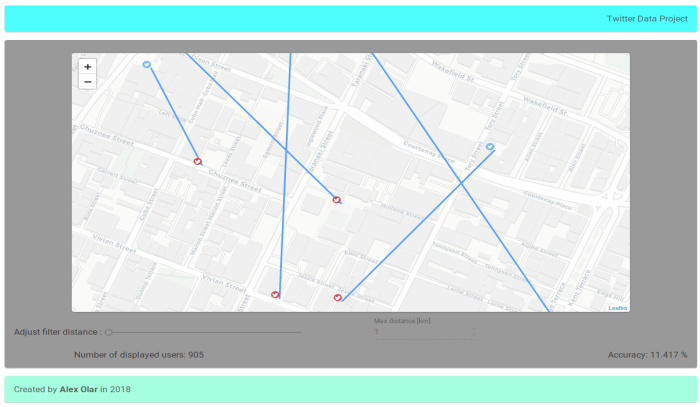


Figure 4: Default value - zoomed

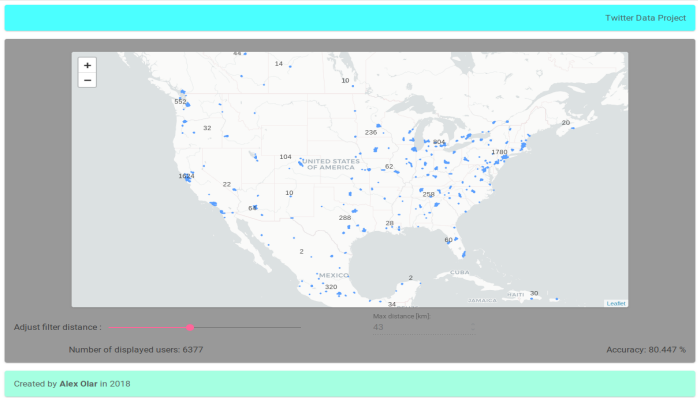


Figure 5: Changed distance - zoomed - USA

IV. Issues

During the week of submission I was only able to connect to the server once. Since I did not save the 5GB of data but waited that 2 minutes each and every time to query it I could not process all the data. My algorithm does not even scale very well, therefore I am not sure if it were possible at all. What I had is around 10 000 users with predicted and actual locations saved with distance in *csv* two weeks ago. However, I consider the project finished since as the data processed is most probably has the same statistics as the rest of it and I processed around 2 % of the total dataset.

V. Overview

Therefore I completed all the tasks required to finish this project. I was not sure whether the visualization tool is enough progress or not. It is not that straightforward to do these things as it seems and there are some very nitpicky moments during the process.

My work can be found on GitHub [2] and on my Heroku [3].

References

[1] LeafletJS. https://leafletjs.com. 2018.

[2] Alex Olar. https://github.com/qbeer/photoz. 2018.

[3] Alex Olar. https://twitter-localization.herokuapp.com. 2018.

[4] scikit learn. http://scikit-learn.org/stable/. 2018.

[5] Wikipedia. https://en.wikipedia.org/wiki/dbscan. 2018.

[6] Wikipedia. https://en.wikipedia.org/wiki/haversine_formula. 2018.