Personal Interaction Studio 2018

Competition in the fight against air pollution

Serge Morel

May 30, 2018

Abstract

Living in 2018, where global warming is a huge topic everywhere, we could really use some competition between cities and/or countries to boost the fight against air pollution. Deforestation is one of the aspects of global warming with the biggest influence, most analysts even rank it above carbon emissions from cars and trucks. Forests, and in general trees, have a direct effect on the carbon dioxide concentration in the air. Taking this even further, studies have shown that greenery can help relief stress, encourage physical exercise and even soothe mental illness. This project will explore the amount of tree growth per city in Switzerland using satellite imagery with the possibility to extent to planet Earth. An interface to visualize and measure the amount of tree coverage in Swiss cities will be provided. Comparison of different cities will also be provided on an interactive map, showing which city outperforms the other in their efforts to reduce global warming. Swiss cities will be ranked with a custom-made scale index. Furthermore, they will receive - previously unmapped - statistics about their overall tree coverage. Already known measurements such as the number of cars, relative population and city area will also be integrated in the city ranking to provide fair comparison.

1 Technical Documentation

1.1 Data Retrieval

As the goal of this design studio is to create a tool which provides measures for tree coverage & density, we need to have a way of collecting data which represents this. Therefore, we take a look at satellite imagery from different kinds of providers. High-resolution imagery does not come for free and after extensive searches I ended up with using MapBox instead of the maybe more popular Google Maps. MapBox provides a really easy to use API for fetching tiles, given their x and y coordinates and even provided me with functions to convert latitude and longitude to this specific coordinate system. No limit in fetching tiles was found as well, so that's a big plus.

Exploring some different kind of zoom levels, I decided to use zoom level 16, which gives us a resolution of $2.387 m/pixel$ with each tile being $256x256$. MapBox also provides functionality to upscale these tiles to better quality, resulting in $512x512$ tiles.

1.2 Labeling

If we want to start using machine learning techniques to detect tree densities of satellite imagery, we need to have training labels because unsupervised learning was not an option considered in this project. Using Overpass Turbo, we could extract existing data for trees and forests. Although this was not exact at all and a lot was not even mapped.

Another option is also manually labeling tiles, but of course that is not really feasible if you need thousands of tiles for machine learning techniques. However, this will end up being feasible in the techniques I used in the final implementation. See section 1.3.

1.3 Machine Learning

Once a solid way of retrieving data was retrieved, it was time to look at machine learning techniques to start predicting tree densities. A lot of options were explored and most of them seemed either unfeasible or would require too much time. This section will list some of the methods and techniques applied as well as their reason why they were skipped or implemented.

1.3.1 Convolutional Neural Network

The first and most obvious way to go about this was using a Convolutional Neural Network, with as input a tile and as output a tree density. So for this I needed tree densities as labels, which I could calculate from the Overpass Turbo data. Unfortunately, this was not a decent data set at all and thus this method did not yield good results.

1.3.2 Fully Convolutional Network

Another approach which would be really nice if I had a proper data set, is using Fully Convolutional Networks (FCN) or U-Nets. This gets a tile as input and outputs an image with the same size, but segmented in categories. In my case there would only be 2 categories: trees and non-trees. Again, this method turned out not useful as I did not have enough data.

1.3.3 Pixel Classifying

So the main problem is the lack of proper labeling data because we need thousands of inputs to properly train any model. That's why I started doing pixel based classifying, because for that I could possibly have way more data than I initially had. If, for example, I label 10 tiles manually, I would already have over 2 million inputs ($512 * 512 * 10$). So that's why I started manually labeling 15 satellite tiles. After balancing out this data set (to have an equal amount of tree/non-tree pixels), I had 768622 input pixels.

The pixels are then transformed to a feature vector of length 10. This includes the pixel's color values in the CIELAB colorspace along with entropy values and illumination invariant values.

With this data I can basically train any basic machine learning model. I tried AdaBoost with 512 decision trees as base classifiers as well as some con-

Table 1: Score features & weights

Feature	**Weight**
Tree density	160
Tree sparsity	240
Cars per inhabitant	30
Relative area	2
Relative population	1

figurations of simple neural networks (up to 3 hidden layers and 256 nodes per layer). These both performed really great and both gave accuracies around 92%.

Of course, this way of working is not really the way a human would detect trees in satellite imagery, the algorithm does not take surrounding pixel classifications into account. That's why after classification, I apply 2 image processing filters called *erosion* and *dilation*. This method effectively removes solely detected pixels which are detected in e.g. an open field and also makes detected areas more resemble tree crowns.

The final prediction was done for all tiles in the cantons Vaud and Genève. This data set is around 20GB.

1.4 Post Processing

After a tile is classified, it is easy to calculate the density. Sparsity of the trees is also calculated because the same density of trees in a less sparse configuration is believed to contribute more to clean air.

With this data we are able to calculate our scores for cities, cantons, random areas, ... Along with the density and sparsity of trees, we also include several other measures to provide fair comparison between cities and cantons. The features are listed in Table 1. The statistics for this were fetched from The Federal Statiscal Office of Switzerland

The prototype also includes viewing statistics for custom drawn areas, for this I do not have all these featues, so the only features I use in that case are tree density, sparsity and relative area.

Because saving each pixel and its coordinate in the database would result in billions of records, I split each tile in chunks of $32x32$ pixels. We then save every tile in the database, with its density, sparsity and polygon. Fetching the data for Vaud and Genève results in a table with over 5 million records.

1.5 Possible Improvements

There is a lot of room for improvement here. For starters, the machine learning model could be a lot better and was note fine-tuned in any way. One could even use predictions made using the pixel classifier as input for, for example a U-Net, as they provide a lot better labels then Overpass Turbo did in the beginning. Furthermore, there could be a lot more parameters taken into account in the score calculation which contribute to air pollution. Another thing to mention is that the size and duration of fetching data is huge. Possibly using U-Nets will improve this and maybe changing the zoom level could help with this as well. Last but certainly not least, more training data. As I only classified 15 tiles, this

is is really region dependent and it would yield a lot better results if the model was trained using randomly selected tiles all over the country. There were quite some errors when regions had to be predicted where the satellite color range was completely different.

2 Code Documentation

The code for this project contains mostly Jupyter Notebooks as they were used to quickly test things in an easy to use environment. However the final implementation of the prototype was done using Django. You can find the GitHub repository at `https://github.com/Vluf/PXS2018`.

2.1 Notebooks

The notebooks were used to extract data, train and test different models, try out functions for drawing polygons and eventually predicting all the tiles for the final prototype. A lot of the first prototypes of functionality was implemented here, as can be seen in the Python files included in this directory on the repository.

2.2 Django

To run the actual prototype (resides in the "PXS" directory), you need a running PostgreSQL service listening on the default port. You will need several python image processing packages, so just go along with installing all these when prompted during running the project.

The polygons for the cities, cantons and countries are included in the repository and can be imported in the database by running a new Django shell (`python manage.py shell`) and then run the `load_*.py` scripts. The `load_data.py` and `calc_*.py` scripts won't work, as you need the actual predicted data for this. For this you will have to run the notebook *Fetcher.ipynb*, which produces pickle files with the predicted tiles for a certain region (default is Vaud).

Running the actual prototype is as simple as running `python manage.py runserver`.

The website is very basic and can be improved a lot layout-wise. The ranking and about page are also not implemented due to time limitations.