"sensormap extension"

Assignment 3

May 14, 2019

1 Data set

In this assignment you will build upon assignment 2 and include an additional sensor type in order to display aggregated data on a map.

Use the trimmed version of the assignment data from assignment 2.[1]. This assignment is restricted only for Graz.

2 Tasks

2.1 Official Measurements of the City Graz and Air Quality (4p)

2.1.1 Preparing and Parsing the Official Sensors

The city Graz has made air quality data from six sensors public.[2] The coordinates of each of the stations are to be taken from the air quality reports.[3]. Use the following detailed search under http://app.luis.steiermark.at/luft2/suche.php to download the required data sets. Use the following configurations: (i) All sensors whose name start with *Graz-* **and** have the components PM10. (PM2.5 will not be needed for this task). (ii) Limit the time period to the timepoint of the first and last values from the old dataset (from luftdaten.info). Extract the data (evtl by Hand) from the .xls file and create the input file. Use the coordinates from the yearly report in order to define the positions of the sensors. Save the extracted data set in a folder named `data` and include it when turning in your assignment!

2.1.2 Air Quality

Calculate the relative air quality for each sensor. Therefore, calculate the average value of each city. Then calculate for each sensor the deviation from this mean (average). The worst sensor (largest negative deviation) should have the value -1 and the best sensor (largest positive deviation) should have the value $+1$. All other values should be proportional to these limits. Save these values in the property named `air_quality`.

Important: Perform this calculation separately for the official and the private sensors!

2.1.3 Package Modifications

Modify your package from the last assignment. The following changes are necessary:

[1] https://github.com/pkasper/info2-bm/blob/master/2019/assignments/assignment_3/sensors_ass_3.csv
[2] http://www.umwelt.steiermark.at/cms/ziel/3611708/DE/
[3] Seite 56ff. http://app.luis.steiermark.at/berichte/Download/Jahresberichte/Jahresbericht_2017_C.pdf

- The class `Sensor` should have the read only Property `official`. This has the value `False` for all old data (from the private sensors) and `True` for the new official sensors data.
- The function `parse_official(data_path)` (in `sensormap_functions`). This function parses the data you have prepared from the official sensors.
- The function `create_bokeh_plot` should check whether values for PM2 exist (these are not present for the official sensors). Plot only PM2 when there are values.
- The class `Sensor` should have the property `air_quality` . Allowed values are floats from -1 to $+1$.

2.2 Voronoi Diagram (6pt)

Add to your class `City` the function `calculate_voronoi(official)`. This calculates a Voronoi-Diagram from all the sensors of the city. The parameter `official` defines whether the diagram should be calculated over the official or private sensors.

For your implementation use the `Voronoi` function from Scipy.[4]. Set the parameter `qhull_options = "Qc"`.

Modify the class `Sensor`. Give a new property named `vor_region`, which by default has the value `None`. The Setter should accept `None` or lists (or numpy Arrays).

Both of the following subtasks build upon one another. First implement the basic version and then modify it for the infinite regions.

2.2.1 Basic

Calculate the Voronoi regions of the sensors. The return value of the function has a number of properties. `point_region` gives you the possibility to associate each region with an input. These feature allows you to match a corresponding sensor to a region. `regions` associates the points to vertices (concrete coordinates). If the region has a value of `-1`, then ignore this region in this step. **Notice:** There is always an empty region. This can be ignored. Save the coordinates of this region (the vertices) in the property of the corresponding sensor.

2.2.2 Infinite Regions

Calculate the vertices of the infinite region. The steps: Iterate through the values of the property `ridge_vertices` (easiest with enumerate). Whenever a value is -1, then this edge goes to infinity. There is always a known point ($\neq -1$). In order to calculate the edge extract the points of the edge (for example, via `vor_diag.points[vor_diag.ridge_points[i]`; `i` is the index of the ridge_vertex.) From these points calculate the mean(via `np.array.mean(axis=0)`). Select the direction of the vector so that it moves away from the center of all of these points. Multiply the vector so that it has the length 0.3 . Save this new point in a dictionary with key `tuple(-1, <index_known_point>)` and the new point as the value (known point + vector).

Whenever you reach -1 when iterating through the regions then approach it like this: Check whether the point has a right or left neighbor. When it has a left neighbor (with index of the point > 1) then add the point from the already created dictionary with the key (-1, index-1). Do the same for a right neighbor wenn $+1$ exists (this time however with the key (-1, index+1)).

Hint: Create a helper function which calculates the length of the vector. **Hint:** The SciPy function `scipy.spatial.voronoi_plot_2d` takes an object of the Voronoi class and draws the regions. Use this to compare your result.

2.3 Plotting (2p)

2.3.1 Layers in Folium

In this task you have to modify the `create_map` function from your last assignment.

[4]https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.spatial.Voronoi.html

In the first step use the Layer from Folium. Create an instance with `folium.map.FeatureGroup(<name>)`. Then add each of the individual components of the layer (via `layer.add_child(new_component)`. Then add the layer to the map (via `folium_map.add_child(layer_item)`). Finally add a control object to the map via `folium_map.add_child(folium.map.LayerControl())`. This will draw a box in the upper right corner and allow the user to display and hide individual layers.

The following layers need to be implemented:

Sensors Official - The official sensors of the city Graz.

Sensors Private - The inofficial sensors (your previous data set).

Overlay Official - the Voronoi Overlay from all official sensors.

Overlay Private - The Voronoi Overlay from all private sensors.

Hint: For this assignment it is allowed when you only give the parameter `plot_list` the city Graz since we only have official sensors from Graz.

2.3.2 Voronoi Overlay

Use the `vor_region` from the `Sensor` class. Draw for every sensor an appropriate region on the map. The color of the region should respond to the value of the property `air_quality` on a scale of red (bad region) to green (best region). Include an `alpha` value in order to make the Overlay slightly see through. Use the function `folium.Polygon` to draw the Voronoi regions.

Hint: Color codes are Hexcodes (z.B., `#FF0000` for red). Set the keyword argument `color` and `fill_color`. To check the transparency you have the parameter `fill_alpha`.

Hint: Individual regions will be very large and reach beyond the city limits. This is a limitation of the algorithm and completely fine!

2.4 Testing (4Pt)

In the file `assignment_3.py` you should use your package. Test the functionality of your expanded package. Explain your thoughts and steps in the comments as well as in the report. Make sure to test what happens when you use invalid values for the parameters.

2.5 Report (4p)

Write a report which should describe your work and findings. Add your report as a PDF file to your assignment when you turn it in. The following components are important:

- How you divided up the work between the members of the team.
- How you prepared the official sensors data.
- what problems did you have you when programming or working on assignment 3.
- How did you test your program.
- Interpretation of the Results.
  - Reliability of the sensor values.
  - Difference between the official and private measurements.
  - Anything else that you think is important to note.

3 Bonus: Combine Neighboring Sensors (5p)

In this task you should combine sensors that are located close to one another. Create a function `combine_sensors(<int>)` in the `City` class. The Parameter ist der minimum distance between sensors. All distances below this value should result in the sensor values being combined. Make sure that your implementation considers/calculates

the sensor values separately (official vs. luftdaten.info). You can decide by yourself how you want to combine the sensor values.

The following thoughts are necessary:

- How do you combine the sensor values?
- What is the position of the combined sensors?
- What happens if a sensor has more than one neighbor?

Explain your thoughts, decisions and findings in the report!
Important: There is no incorrect solution!

4 Restrictions

- Do not add any bloat to your submission
- Use built-in functions where possible
- Only use/import explicitly allowed packages
- Do not use any command line arguments
  - You may use extra parameters. However, your program should also work without their use!

4.1 Allowed Packages

- `os`, `sys`, `math`
- `numpy`, `scipy`, `pandas`, `tqdm`
- `matplotlib`, `seaborn`, `bokeh`, `folium`, `geopy`

5 File Headers

All Python source files have to contain a comment header with the following information:

- Author: – Your name
- MatNr: – Your matriculate number
- Description: – Purpose of the file
- Comments: – Any comments as to why if you deviate from the task or restrictions

Please copy and paste the example and change its contents to your data. (Depending on your PDF-viewer you may have to fix the whitespaces!)

Example Header:

```
#############################################################################
# Author:      [FIRST NAME] [LAST NAME]
# MatNr:       [MAT NR]
# Description: [SHORTDESCRIPTION]
# Comments:    [ANY RELEVANT COMMENTS.
#              CAN BE MULTILINE]
#############################################################################
```

6 Coding Standard (Deduction up to 50 percent!)

This lecture follows the official PEP 8 Standard[5]. it defines basic formalities as to how your code should look like. In particular, please look at the following aspects:

Language. Programming is done in English. This is to ensure someone else at the other end of the world can read your code. Please make sure all sources you submit are thus written in English. This covers both variable names and comments!

Spaces, not tabs. Indent your files with 4 spaces instead of tabs. PEP 8 forces this in Python 3 (whilst allowing more freedom in Python 2). Most editors have an option to indent with 4 spaces when you press the tab button!

Descriptive names. Use descriptive names for variables where possible! Whilst a simple i can be enough for a simple single loop code can become very messy really fast. If a variable has no purpose at all, use a single underscore (_) for its name. Should you be uncertain if a name is descriptive enough, simply as a comment describing the variable.

120 **characters line-limit** PEP 8 suggests a maximum line length of 79 characters. A different established limit proposes 120 characters. In this lecture, we thus use the wider limit to allow for more freedom. The maximum number of characters is 120 including indentations! Note that this is also applies to comments!

7 Automated Tests

Your code will be tested by an automated program. Please make sure your submission follows all the restrictions defined in this document. In particular concentrate on the predefined names for functions and variables!
If your submission fails any of the automated tests, we will look into your code to ensure the reason was not on our end and to evaluate which parts of your code are correct and awards points accordingly.

[5] `https://www.python.org/dev/peps/pep-0008/`

8 Submission

8.1 Deadline

June 2nd 2019 at 23:59:59.

Any submission handed in too late will be ignored with the obvious exception of emergencies. In case the submission system is globally unreachable at the deadline it will be pushed back for 24 hours. If for whatever reason you are unable to submit your work, contact your tutor *PRIOR* to the deadline.

8.2 Uploading your submission

Assignment submissions in this course will always be archives. You are allowed to use .zip or .tar.gz formats.
In addition to your Python source files, please also pack a *readme.txt*. The file is mandatory but its contents are optional. In this file please write down how much time you spent on the assignment and where you ran into issues. Your feedback allows for immediate adjustments to the lecture and tutor sessions.
Upload your work to the Palme website. Before you do please double-check the following aspects:

- File names and folder structure (see below)
- Comment headers in every source code file
- Coding standard upheld

8.3 Your submission file

```
assignment_3.zip (or assignment_3.tar.gz)
    assignment_3.py
    readme.txt
    report.pdf
    data
        [Your official sensors data]
    sensormap
        __init__.py
        [All files in your submission(.py)]
```