This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 776804 — H2020-SC5-2017.

NEXT
New Exploration Technologies

DELIVERABLE 4.11
Appendix 2
Technical Specification: *GisSOM*

Horizon 2020 Project: **NEXT**

Author(s): **Sakari Hautala, Johanna Torppa, Jaakko Madetoja**

Institution: **Geological Survey of Finland**

Date: **19.11.2020**

TABLE OF CONTENTS

LIST OF FIGURES

1 INTRODUCTION

The purpose of this document is to describe the software design, class diagram and the testing procedure of *GisSOM* that is one component of the software implemented in the European Union funded H2020 project NEXT.

The core of the NEXT software (Figure 1) is the nextsomcore (D 4.11 Appendix 1), which applies self-organizing maps (SOM) and k-means clustering for analyzing multivariate data. nextsomcore can be utilized using three different graphical user interfaces that allow handing of spatial data: ArcGIS (D 4.13), advangeo® (D 4.12) and GisSOM (this document). ArcGIS and advangeo® are commercial software, while GisSOM is freeware and open source.

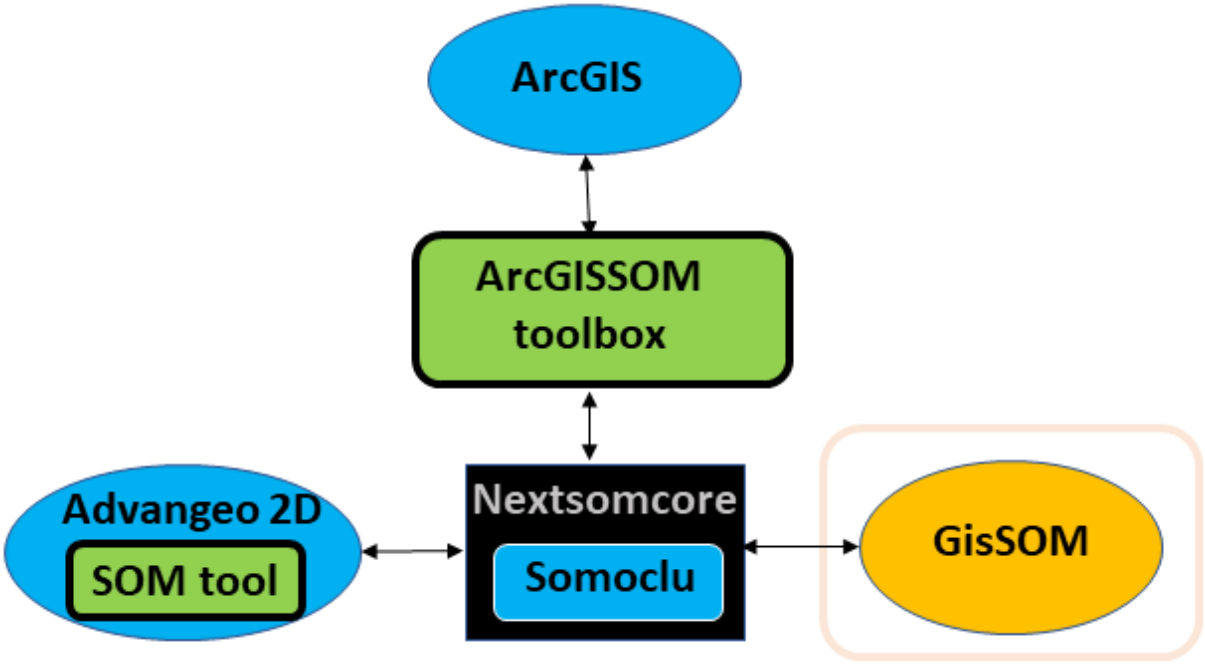

Figure 1. ***Structure of the self-organizing maps software developed in the NEXT project. Blue color refers to existing software, while green, orange and black components were implemented in NEXT. The GisSOM component is described in this document***

2 SOFTWARE DESIGN AND CLASS DIAGRAM

GisSOM is implemented in C# using the Windows Presentation Foundation (WPF) framework, according to the Model-View-ViewModel (MVVM) design model. Computational tasks related to data preprocessing and visualization of the results are implemented as Python scripts. The software uses Python version 3.7, and the Python scripts and the python environment are bundled into executables using PyInstaller.

The following Python packages/libraries are used:

- somoclu

- matplotlib
- numpy
- seaborn
- pandas
- gevent
- dash
- plotly
- GDAL
- flask
- scipy
- scikit-learn

Figure 2 presents the Model, View and ViewModel classes. The Model class handles all the data, the View classes handle the user interface and the ViewModel classes act as an interface between these two.

SomModel contains all the parameters that are used in data preprocessing as well as in SOM and k-means computations. It also contains links to the input data files and output files. The **MainViewModel**-class handles transitions from one view to another. **SomViewModel** handles all the rest of the UI logic and acts as a link between the View and Model classes. The **SomTool**-class is used to launch all Python processes.

All Views are user controls that are hosted are hosted in **MainWindow**. Input data is selected and prepared in the **DataPreparationView** (log transform, winsorizing, attribute selection). SOM computation parameters are selected in the **SomParameterView**. Results are shown in **SomResultMenuView**, which hosts six tabs: **SomSpaceResultView** (attribute maps and U-matrix in SOM space), **GeoSpaceResultView** (input data attribute, q-error and k-means cluster maps in geospace), **ClusteringView** (Running k-means clustering again after running SOM, and selecting clustering), **BoxPlotView** (boxplots of data distributions within the k-means clusters), **ScatterPlotView** (attribute correlation plots in 2D), and **InteractiveView** (Interactive plotting, currently contains SOM clusters plot that allows you to select individual cluster and plot it in geospace).

The **TabViewModel**, **TabWindow** and **TabContent** classes are a smaller framework (using the Dragablz package) that makes it possible for the user to drag result tabs into a separate window.

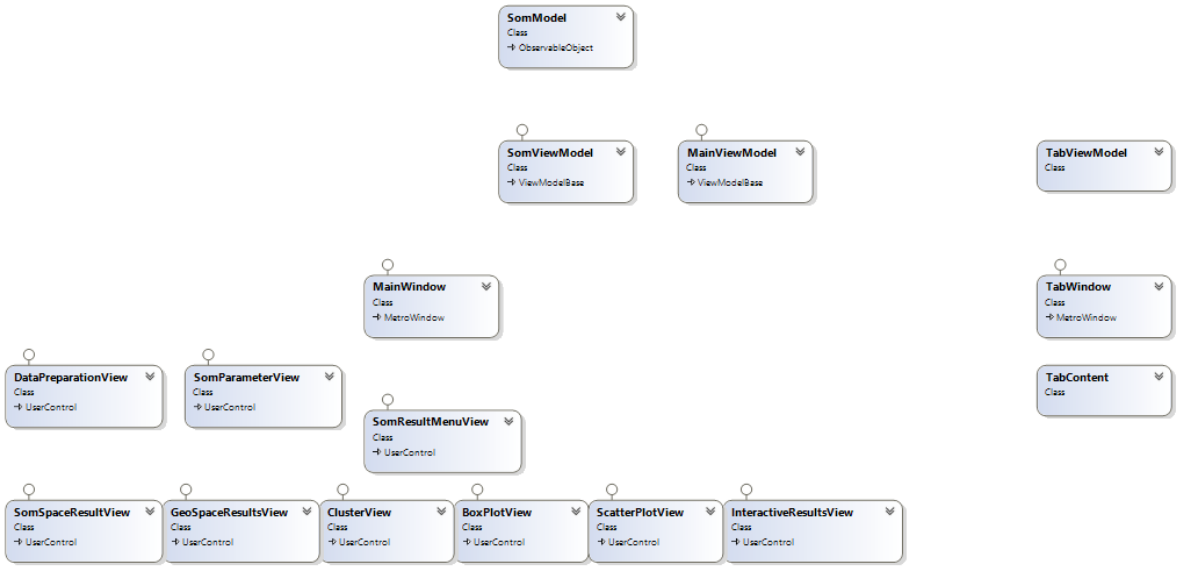

Figure 2. Model, View and ViewModel classes.

Figure 3 presents the App, MainWindow and smaller service classes. Class **App** serves as the launching point for the application. Views use the **ViewModelLocator** to access the ViewModels. **MyInterTabClient** serves the same function as the 3 other Tab-classes presented in Figure 2. The **DialogService**, implementing the **IDialogService** interface, is used to open file browser dialogs. The **ValueConverters** are simply small value converter service classes, required for presenting data in a

different form in the Views while still adhering to the MVVM principles. **Settings** contains general application settings, and **Resources** contains external resources (fonts, etc.).

Figure 3. App, MainWindow and smaller service classes.

Figure 4 presents the Python scripts that perform the computational tasks related to data preparation, and visualization of SOM and k-means results. SomViewModel calls all the Python scripts. The workflow and order of execution for the Python scripts is illustrated by the blue lines, but the line is dashed because there is no actual direct connection between the scripts (this is handled by SomViewModel). The script *split_to_columns* splits the input data file to individual columns that are saved as binary 2D numpy arrays. These individual columns are used by *edit_column* and *draw_histogram* scripts. *edit_column.py* is used to do the data preparation procedures (winsorize, log transform, etc.), and *draw_histogram* draws a histogram of the selected data column. If the number of datapoints exceeds five thousand points, a random sample of 5000 is taken (with numpy's random.choice function) for plotting the histogram, to speed up the process. After editing the data, the *combine_to_lrn_file* script is used to combine the individual columns back to a LRN file. To draw the som and geospace plots after SOM calculation, *nextsom_plot* is used. The script *cluster_draw* is used to draw the 3 best clustering results. The *nextsom_plot_dash* script draws the interactive clustering plot, and when a cluster is selected *nextsom_plot_dash_draw* script is called to draw the selected cluster in geospace.

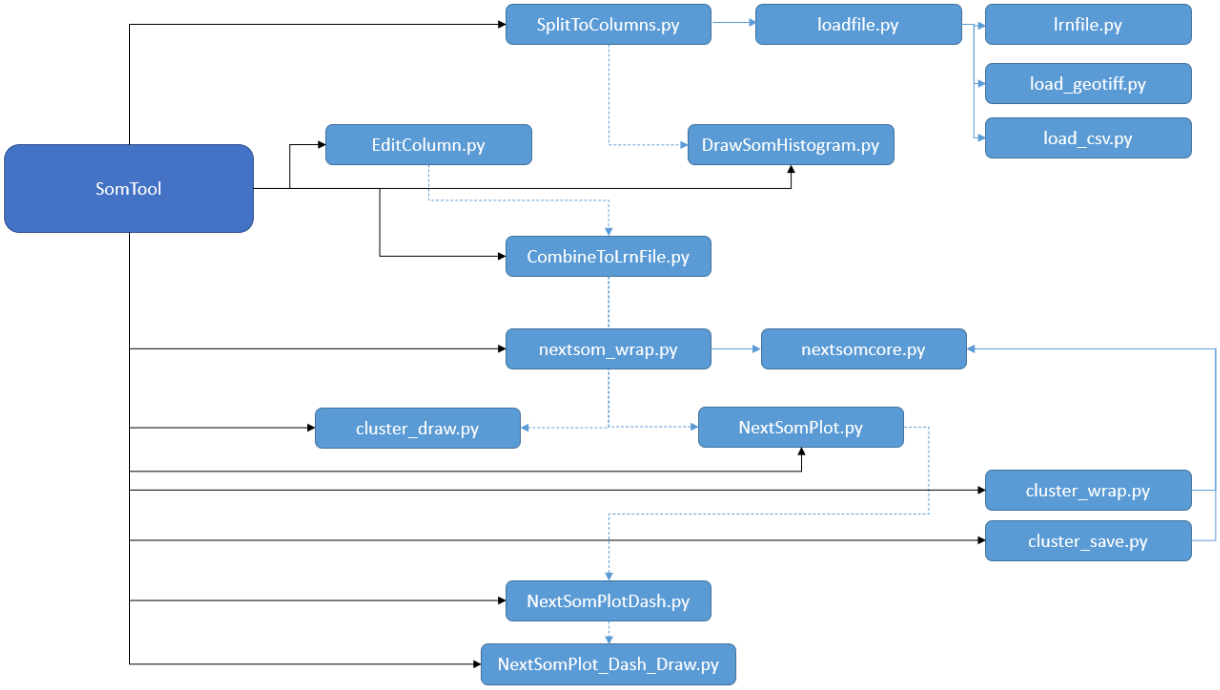

Figure 4. Python scripts for computational tasks related to data preparation and plotting. Solid arrows indicate that a script is called, dotted arrows indicate a logical progression step. Also, the scripts are arranged roughly top-down in order of execution.

3 FUNCTIONALITY IN CLASSES

3.1 Data preprocessing

Normalization

Normalization is done to transform the data to a user defined range [x_{min},x_{max}] using the following formula for the rescaled value r:

$$\boldsymbol{r} = \frac{(r_{max} - r_{min}) * (\boldsymbol{x} - x_{min})}{x_{max} - x_{min}} + r_{min}$$

where $\boldsymbol{x}$ is the vector of the untransformed values.

Winsorizing

Winsorizing means assigning a limiting value to parameter values below and above the given limiting value.

Log transform

Logarithmic transform is carried out by first shifting all the parameter values to the positive range. Then, a natural logarithm is applied to the values and computation is performed using the

transformed values. The log transform is mutually exclusive with winsorizing, as they are both performed on the original data values.

3.2 Self-organizing maps and k-means method

SOM and k-means computations are carried out using the somoclu package (Wittek et al., 2017). After the initialization of the SOM neuron weights, the training of SOM utilizes competitive learning (Kohonen, 2001): For a given data point, the neuron with the smallest Euclidean distance is found; this neuron is called the best matching unit (BMU). The weights of the BMU and the neurons close to it are updated to be closer to the data point. The formula for updating the weights is

$$w(t+1) = w(t) + \alpha(t)h(t)(x(t) - w(t))$$

where $w(t+1)$ is the new weight for a given neuron, $w(t)$ is the old weight, $\alpha(t)$ is monotonically decreasing coefficient (learning rate), $h(t)$ is a neighborhood function, and $x(t)$ is the input data value. The learning rate ensures that the area in which the weights are updated shrinks over time and the neighborhood function ensures that the update is smaller the farther away the neuron is from the BMU in SOM space.

After SOM calculation, k-means clustering can be applied to its neurons. The algorithm is iterative; it assigns observations to the closest cluster centroid and recalculates the centroids. This is repeated until no updating happens.

The user provides the minimum and maximum number of clusters for which k-means clustering is tested. As the initial random assignment of clusters affects the results of the algorithm, k-means is run multiple times (user provides the number of initializations) for each number of clusters in the given range. The best clustering result for each number of clusters is saved, and the three best clustering results are shown in the user interface and stored. The goodness of clustering is measured using the Davies-Bouldin index (Davies & Bouldin, 1979).

3.2.1 Quantization error

The quality of SOM is usually measured using two quantities. The *topological error* describes how closely similar data vectors are located on SOM and the *quantization error* is a measure of the goodness of clustering of data vectors in each SOM cell. In GisSOM, computation of only quantization error is implemented so far.

The quantization error is computed using the equation

$$E_q = \frac{1}{N}\sum_{i=1}^{N} \|\boldsymbol{X}_i - \boldsymbol{BMU}_i\|,$$

where N is the total number of data vectors, $\boldsymbol{X}_i$ is the i^{th} data vector and $\boldsymbol{BMU}_i$ is the SOM codebook vector in the best matching unit of $\boldsymbol{X}_i$.

3.3 Results

The results of the som calculation are saved into .CSV and .txt files, separate files for results in SOM space and geospace. Also in the case of GeoTIFF input, the som results
The results are plotted, and the plots are divided into five separate tabs in the software: Somspace results, Geospace results, Boxplots, Scatterplots and Interactive results.

3.3.1 Somspace results

Somspace plots are heatmaps, representing the following SOM parameters:

1. value of each codebook vector element (corresponding to data parameters)
2. u-matrix (the magnitude of the difference of the codebook vectors in neighboring SOM cells)
3. K-means cluster. The left-most numbers on the color bar represent the number of the cluster, and the numbers on the right represent number of data points per cluster.
4. number of data points clustered in each SOM cell

Depending on selected grid type, the heatmaps have either square or hexagonal cells. The square cell variant uses the library 'seaborn' for heatmaps, while the hexagonal plots use a custom plotting function, that uses a scatterplot as its base. Both seaborn and the custom function are based on 'matplotlib'.

If the labeling option is used, labels will be shown on the map as numbers. A separate legend plot will be drawn, that contains the labels or combinations of labels represented by each number, so that labels ending up in the same SOM cell can be distinguished from one another. New labels can also be added via the appropriately named button in the Somspace results view, this will draw a separate plot with the new label data.

3.3.2 Geospace results

Geospace result images show the k-means clustering results, best-matching unit codebook vectors and quantization errors in geographical space. Used libraries depend on whether the input data is grid or scatter type: Grid type uses seaborn heatmaps, while the scatter type uses the same plotting function that is used to draw the hexagonal heatmap plots.

3.3.3 Boxplots

These images different SOM result data parameters of the k-means clustering results as boxplots. The Seaborn library is used to create the plots.

3.3.4 Scatterplots

The scatterplots have two variables cross plotted: Each point in the scatterplot represents the SOM data point in x,y-coordinates with one parameter as x and another as y. Seaborn is used for drawing the scatterplots.

3.3.5 Interactive plot

The interactive results -tab has a somspace cluster plot on the left side, and a user-specified geospace parameter plot on the right side (using original data points). The left plot is a html page hosted in an embedded web browser (CefSharp Chromium browser), while the right-side plot is an basic image file. By left-clicking on any cell on the somspace plot, the corresponding cluster or som cell will be highlighted on the geospace plot on the right side.

The base of the interactive plot is a local html web page created with dash, hosted in localhost. The libraries 'flask' and 'gevent' are used for the hosting, while the library 'dash' is used to create the html page itself.

When the mouse cursor is hovering over the plot, the som coordinates are displayed as hoverdata (tooltip), and when the plot is clicked, the som coordinates and cluster number are saved as click data. Communication from the dash plot to the C# side of things is handled by writing the click data to a text file in the result folder. An instance of the FileSystemWatcher class is created in the C# side and set to monitor changes in the clickdata text file. When the file is changed, a process is launched to draw the geospace result plot based on the selected values.

The dash page has an automatic shutdown timer of 10 minutes, but if a new plot is created or the program is closed, a http post request for shutting the plot down will be sent from the C# side.

4 REFERENCES

Deliverable 4.11 Appendix 1: Technical Specification – *nextsomcore*

Deliverable 4.12: advangeo 2D with SOM®

Deliverable 4.13: ArcGISSOM toolbox

Kohonen, T. (2001) Self-Organizing Maps. Springer-Verlag.