

**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**

*Disclaimer*

*The information in this document is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information as its sole risk and liability. The document reflects only the author’s views and the Community is not liable for any use that may be made of the information contained therein.*

TABLE OF CONTENTS

[1 Introduction 3](#_Toc7529733)

[2 Software Design and Class Diagram 4](#_Toc7529734)

[3 References 6](#_Toc7529735)

List of Figures

[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. 3](#_Toc7529736)

[Figure 2. Model, View and ViewModel classes. 5](#_Toc7529737)

[Figure 3. App, MainWindow and smaller service classes. 5](#_Toc7529738)

[Figure 4. Python scripts for computational tasks related to data preparation and plotting. 6](#_Toc7529739)

# 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.

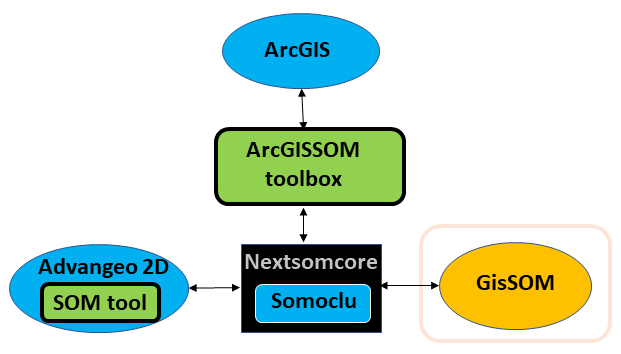


Figure 1. Structure of the self-organizing maps software developed in the NEXT project. Blue color refers to existing software, while green, orange and black components were implemented in NEXT. The GisSOM component is described in this document

# Software Design and Class Diagram

GisSOM is implemented in C# using the Windows Presentation Foundation (WPF) framework, according to the 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.

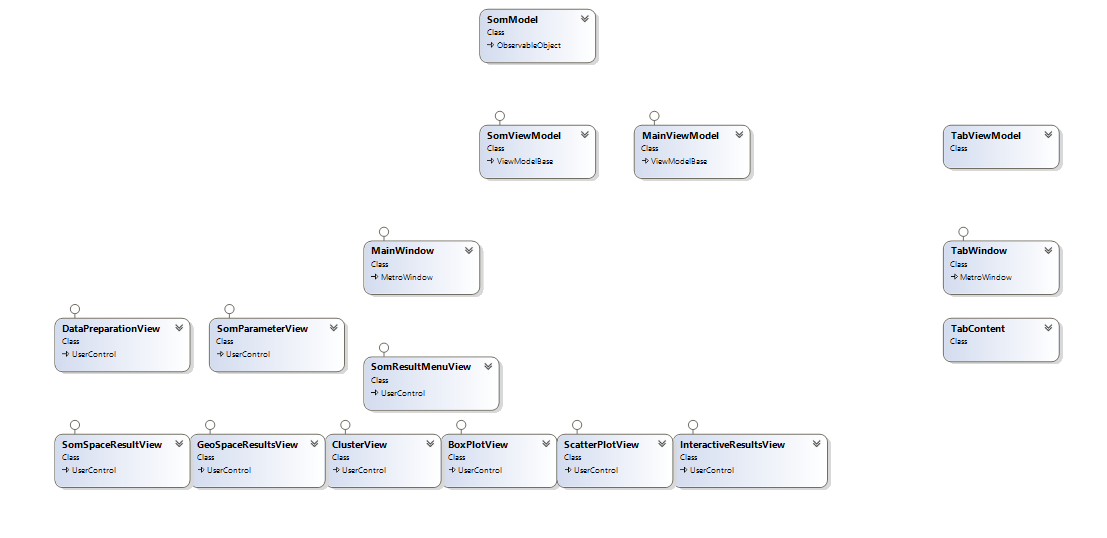


Figure 2. Model, View and ViewModel classes.

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.).

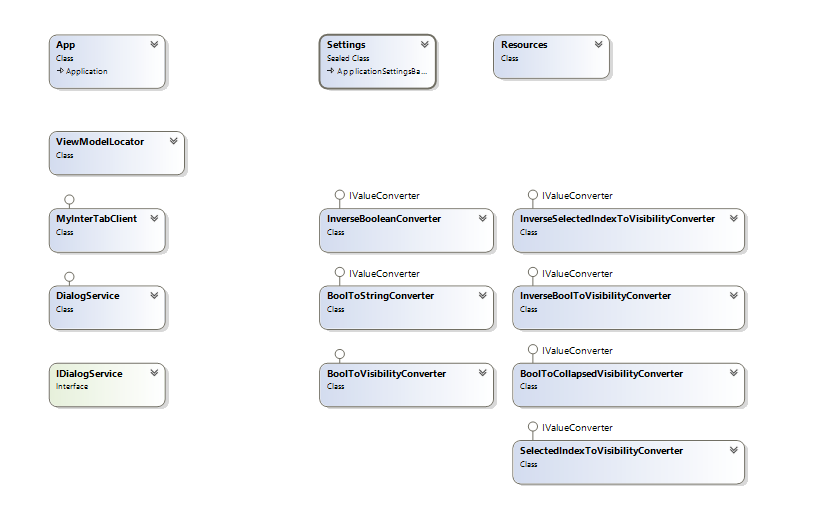


Figure 3. App, MainWindow and smaller service classes.

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). SplitToColumns splits the input data file to individual columns that are saved as binary 2D numpy arrays. These individual columns are used by EditColumn and DrawSomHistogram scripts. EditColumn is used to do the data preparation procedures (winsorize, log transform, etc.), and DrawSomHistogram draws a histogram of the selected data column. After editing the data, CombineToLrnFile script is used to combine the individual columns back to a LRN file. NextSomPlot is used after SOM calculation, to draw the maps, scatterplots and boxplots. cluster\_draw is used to draw the 3 best clustering results. NextSomPlotDash script draws the interactive clustering plot, and when a cluster is selected NextSomPlot\_Dash\_Draw script is called to draw the selected cluster in geospace.

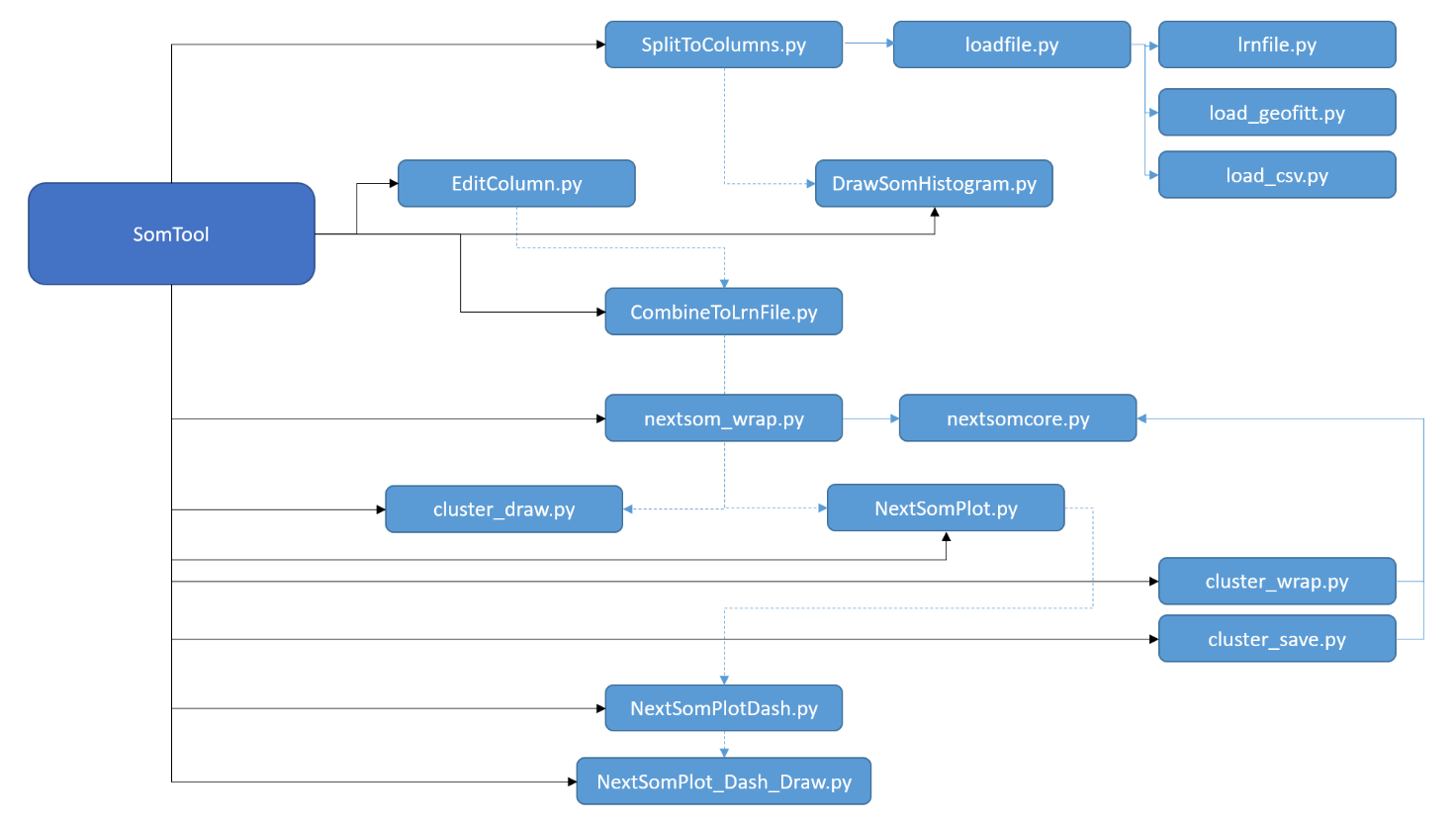


Figure 4. Python scripts for computational tasks related to data preparation and plotting. Solid arrows indicate that a script is called, dotted arrows indicate a logical progression step. Also, the scripts are arranged roughly top-down in order of execution.

# Functionality in classes

## 3.1 Data preprocessing

**Normalization**

Normalization is done to transform the data to a user defined range [*xmin,xmax*] using the following formula for the rescaled value :

where 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.

## 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

where is the new weight for a given neuron, is the old weight, is monotonically decreasing coefficient (learning rate), is a neighborhood function, and 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).

### 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

where N is the total number of data vectors, ***X****i* is the *i*th data vector and ***BMU****i* is the SOM codebook vector in the best matching unit of ***X****i*.

## Results

* Scatterplots are created using the SOM codebook vectors
* Boxplots are created using the original data set

# 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.