Exercise: Basics of Self-Organizing Maps (SOMs)

Ellen-Wien Augustijn

December 2016

Contents

Exercise: Basics of Self-Organizing Maps (SOMs)

Self-organizing maps are an unsupervised machine learning technique that can be applied within the Health Domain in two groups of studies:

a. Classification of data containing a large number of variables (e.g. [1])
b. Analyses of time series where the variables are representing time steps (e.g. [2])

Although Self-Organizing maps are not a spatial analytical technique, they can be applied to spatial data when the input variables are linked to a spatial location (for example data collected for a number of administrative units). They are particularly useful when the datasets are large (many variables and many administrative zones).

When applied as a classification technique they differ from more traditional techniques by the fact that they do not require the user to identify the number of classes that should be found, and by the fact that they are un-supervised.

Results of Self-Organizing Maps can be easily mapped back to a GIS to show the result by means of a map.

In this exercise we will start with evaluating an input dataset to understand the structure of the input data. We will then perform a SOM analysis to learn to understand the basic principles, followed by a step in which we will map back the result to a GIS.

1. Understanding the input data structure

Rwanda is the first country where performance-based financing (PBF) was introduced nation-wide. Among other approaches to overcome supply-barriers for health care there is the Performance-Based Financing (PBF) of the health care system [3], based on the belief that health workers are motivated by financial incentives and will try to maximize incomes [4]. PBF is an approach in which it is tried to provide the (often poorly paid) health care providers some incentive to give more or better care [5]. Although research has proven the effectiveness of PBF, more research to the working mechanisms is needed.

One of our students studied this PBF system, by looking at the existence and diffusion of strategies over health facilities. We now make a small step by checking if there are spatial differences between these health facilities that can also have played a role in the choice of strategies.

Rwanda is divided into population cells as can be seen in the picture below (Figure 0-1). In total there are 2147 cells. For these population cells we calculated several indicators of “remoteness” and access to health care for every population cell in the country. We created the following variables:

- Euclidean Distance from the centroid of the population cell to the nearest district hospital

- Euclidean Distance from the centroid of the population cell to the nearest health facility of any type.
- Euclidean Distance from the centroid of the population cell to Kigali (capital of Rwanda)
- Euclidean Distance from the centroid of the population cell to the closest main road
- Euclidean centroid cell to any road
- Sum_ROAD_Length: sum of the length of all the roads in the population cell
- SUM ROAD LENGTH PER AREA: Sum_ROAS_Length/area unit

You should note that some of these variables indicate remote areas when they are large, for example when the distance to the capital city Kigali is large the area is regarded to be remote. Other variables work in an opposite way. When the sum of the road length is high, this may indicate an area with a very well developed road network and this is normally not the case for remote areas. The same also applies for the sum of the road length per area unit.

Your input for this exercise was prepared in Excel as a spreadsheet but it can also come from a GIS, for example the attribute table of a Shapefile. The Excel table was stored as a .csv file and this is the only input you will be using during the exercise.

The general structure of the data is by population cells (admin unit). The columns in your dataset represent the variables the rows in your dataset represent the population cells.

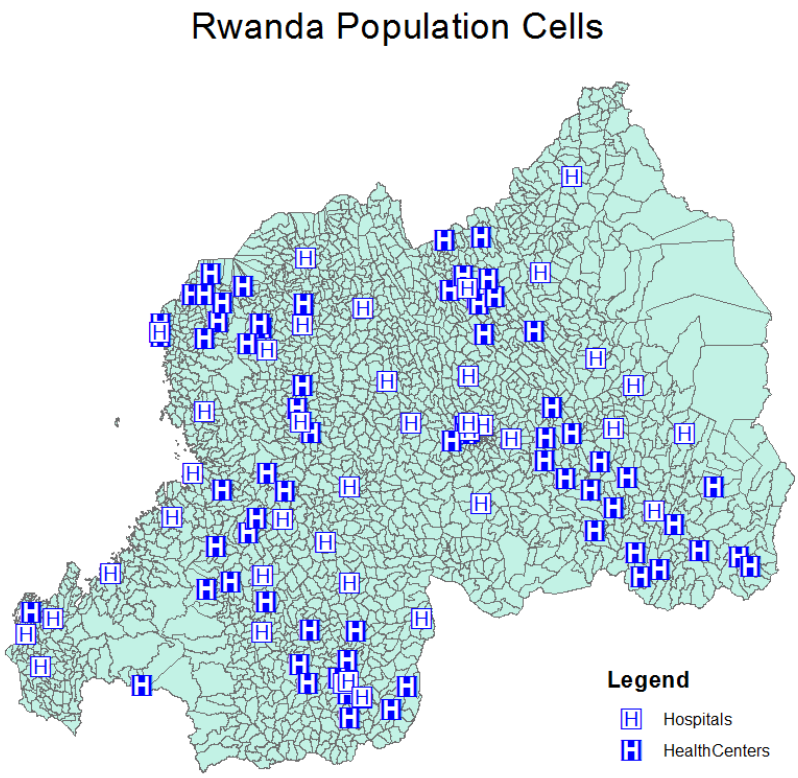


Figure 0-1 Population cells, hospitals and health centers

Besides the .csv file that you will be using during the SOM part of the exercise, you are also provided with a Shapefile showing the population cells and two Shapefiles showing the hospitals and Health Centers. These files will be used at the end of the exercise for visualization purposes.

2. Training a self-Organizing Map

2.1. Installing the Kohonen package

In this exercise we are going to use the R software in combination with the Kohonen package. More information on the Kohonen package can be found in the manual and instruction paper ([6]). In order to follow the instructions you will need an installation of R on your computer. You already installed this software on day 1 of this course. You will also have to install the Kohonen package. If you have not done so yet, you will do this now:

1. Open R
2. SelectCRAN mirror
3. In the Main Menu select “Packages” – “Install package” – “Kohonen”

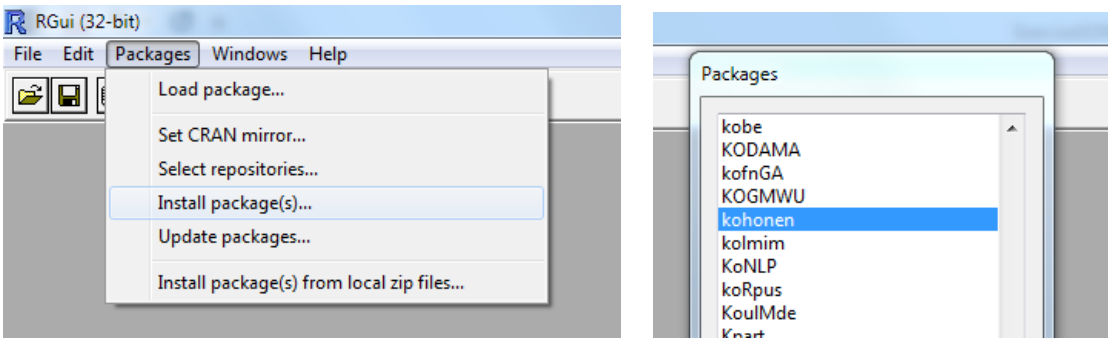


Figure 0-2Installing an R package

There are several packages within R that can perform Self-organizing Maps. It is not so that this package is better compared to any of the other available packages. They all have their own advantages so before you start your research you may want to explore some other packages also. You are now ready to perform your first SOM analysis.

Note: The CRAN mirror is a distribution site for software. Generally the mirrors will all have copies of the software (libraries, source code, documentation) identical to those available from the 'main' download site for the software. This allows you to choose a download site close to you for better bandwidth/latency, balances load across many servers, and provides redundancy so you can always obtain the software even if one or more of the mirror sites is offline.

2.2. Loading package and data

Everything we are going to do within R we will do via a script. This is an easy way to store your workflow and makes it easy to reproduce what you have done or make small adjustments. You can open a new script by going to the main menu and select: “**File**” – “**New Script**”.

Please note that all the R commands, set in `courier` font, can be copied from this document and pasted in the script.

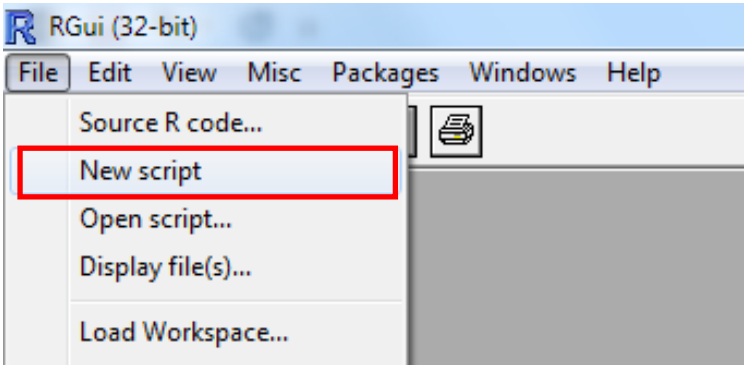


Figure 0-3 Creating a New Script

The first thing we need your script to do is to load the Kohonen package. You will do this by typing the following statement in your script:

```
library("kohonen")
```

This will ensure that the Kohonen package will be loaded.

The second this you would like to do is reset the working directory to the folder in which your data are stored. You can do this by typing the next step in your script:

```
setwd("D:\\EAUN\\Refresher_Course\\face-to-face\\Data_SOM")
```

Make sure that you replace the name ***D:\\EAUN\\Refresher_Course\\face-to-face\\Data_SOM*** with the path to your own directory.

We will now load the dataset into R.

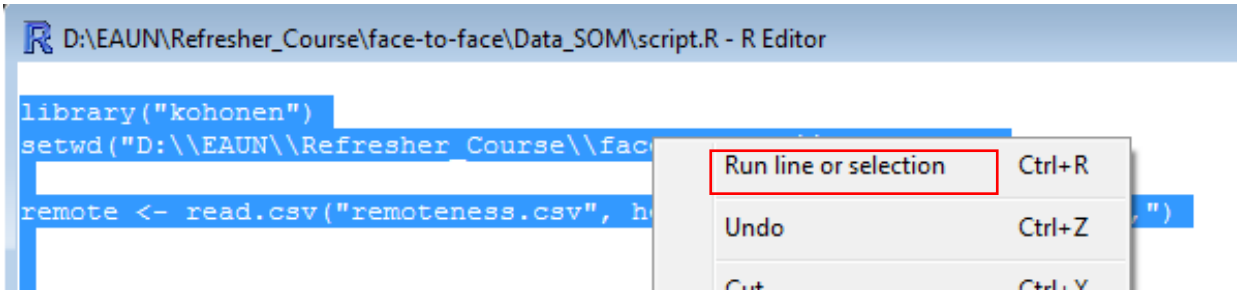
```
remote<-read.csv("remoteness.csv", header=TRUE, sep = ";", dec= ",")
```

You may notice that this command expects the .csv file to have a header and it uses commas to indicate decimals.

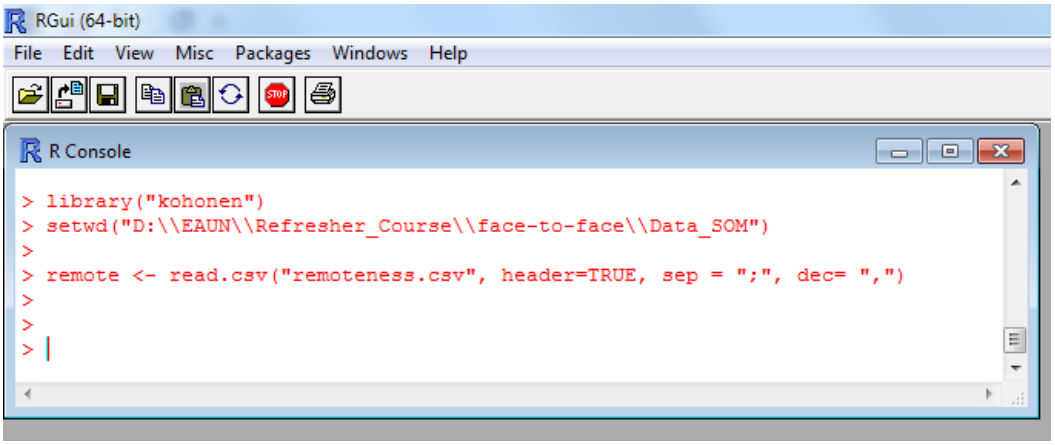
Your script has now three lines:

```
library("kohonen")
setwd("D:\\EAUN\\Refresher_Course\\face-to-face\\Data_SOM")
remote <- read.csv("remoteness.csv", header=TRUE, sep = ";", dec= ",")
```

To test if your script is reading your dataset you can select all three lines, use your context menu to display the options and select "**run line or selection**" from the dropdown menu.



You should see that in your console the following text appears:



This means that your script has read the input data set. You can test this by typing **remote** into your console.

We want to make sure that your data is in matrix format. You can do this adding the following text to your script and run this line:

```
remotemat=as.matrix(remote)
```

2.3. Training your SOM

Before we start to train the SOM we will scale the data values between 0 and 1, and we will set a seed for generating random numbers to ensure that the experiments are repeatable. After these steps, we are ready to train our SOM lattice. This is done by presenting random samples from the training dataset to the Lattice and updating the values.

Scaling your data

The variables in this dataset differ a lot in type and values. The total sum of the length of all roads can be a very large number compared to the distance to the nearest main road. To make sure that all variables are in range we will all scale them in the same way.

Add the following line to your script:

```
remotemat.sc<-scale(remotemat)
```

If you want to see the result you can again plot the object by typing **remotemat.sc** in your console to see the new values. As you see below these values should range between 1 to -1. They are scaled around 0.

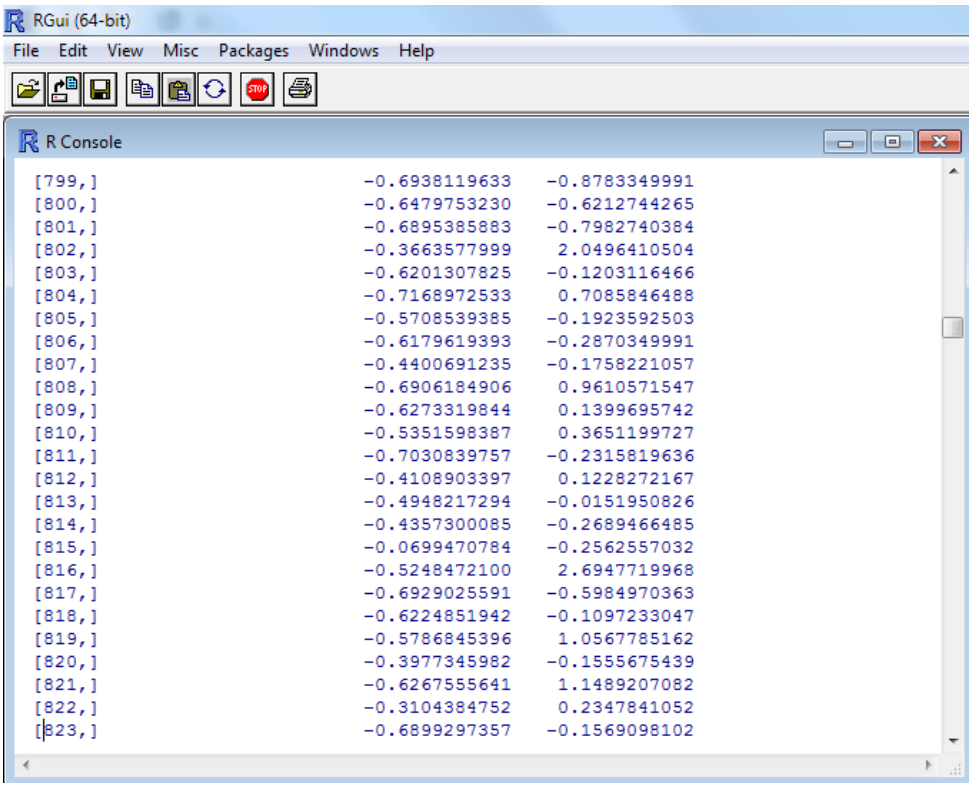


Figure 0-4 Scaled values

Setting seed

Set the seed of R's random number generator, which is useful for creating simulations or random objects that can be reproduced. When you run your script again the random numbers are the same.

Add the following line to your script and run this line:

```
set.seed(7)
```

Training SOM

During the MOOC you learned that the SOM process consists of two steps, training the SOM lattice and mapping back data. We are now going to perform the first step.

Add the following line to your script:

```
remotemat.som<-som(data=remotemat.sc,
grid=somgrid(6,6,"hexagonal"),rlen=100000, keep.data=TRUE)
```

What you see is that we are using the scaled dataset (remotemat.sc) as input data, we will create a somgrid of 6 by 6 rows/columns, so we will have 36 neurons. The parameter *rlen*, defines the number of iterations in the training phase. In this case we will use 100,000 but for larger datasets you want to go higher.

Select the lines in your script and run the code. The training may take some time so wait until it is completely done. Then you can plot the result by adding the following lines to your script:

```
plot(remotemat.som)
```

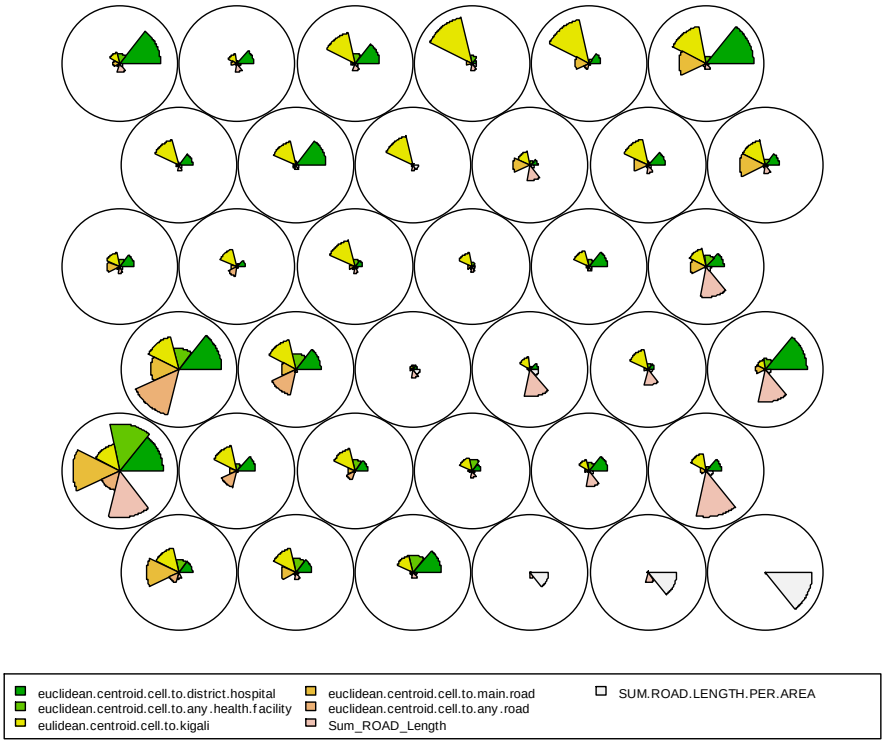


Figure 0-5 Trained lattice

Take a few moments to try to understand the results of this lattice. At the bottom right side of the lattice we see three neurons with small distance to health facilities, Kigali and main roads. The sum of the total road length is not very large but when measured per area unit (km2) then we see that these areas have high number. These neurons do not represent remote areas, they rather represent the opposite (highly accessible areas).

When we look at the left bottom of the lattice, we see neurons that have high values for all variables that express distances to health centers, hospitals, main roads and Kigali. These areas can be regarded as highly remote.

2.4. Visualizations for SOMs

Mapping data onto SOM lattice

After training the SOM lattice, we can map a dataset back onto this trained lattice. This mapping dataset can be the training dataset, a subset of the original training dataset (for example one year of data) or a completely different dataset.

We will again add a line to your script:

```
mapping <- map(remotemat.som,remotemat.sc)
```

In this case we will map the scaled (remotemat.sc) dataset back onto the trained SOM (remotemat.som).

We can see the result by adding another line to the script:

```
plot(remotemat.som, type="mapping", pch =1, main="all")
```

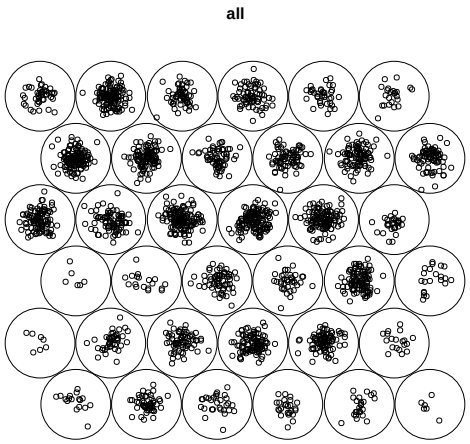


Figure 0-6 Mapping back data onto trained lattice

Every dot (circle) that you see in the lattice is one of your record. You see that some of the neurons have many records mapped and others have very few. Each neuron has a number. This numbering starts at the bottom left corner of the lattice and goes row by row. The top right hand corner has the number 36.

Instead of showing the number of dots in each neuron we can also use color to indicate how many dots each neuron has. The following types of plotting exist: *"codes", "changes", "counts", "dist.neighbours", "mapping", "property", "quality".* The previous visualization we used mapping, but we will now

Several different types of plots are supported:

> ***"changes":*** *shows the mean distance to the closest codebook vector during training.*
>
> ***"codes"*** *: shows the codebook vectors.*
>
> ***"counts"*** *:shows the number of objects mapped to the individual units. Empty units are depicted in gray.*
>
> ***"dist.neighbours"****: shows the sum of the distances to all immediate neighbours. This kind of visualisation is also known as a U-matrix plot. Units near a class boundary can be expected to have higher average distances to their neighbours. Only available for the "som" and "supersom" maps, for the moment.*
>
> ***"mapping"****: shows where objects are mapped. It needs the "classif" argument, and a "labels" or "pchs" argument.*
>
> ***"property"*** *: properties of each unit can be calculated and shown in colour code. It can be used to visualise the similarity of one particular object to all units in the map, to show the mean similarity of all units and the objects mapped to them, etcetera. The parameter* `property` *contains the numerical values. See examples below.*
>
> ***"quality"****: shows the mean distance of objects mapped to a unit to the codebook vector of that* unit. The smaller the distances, the better the objects are represented by the codebook vectors.

As you can see in the list above it would be interesting to show the number of items mapped to each neuron as a count. We can do this via the following line of code:

```
plot (remotemat.som, type="counts", pch = 1, main="count")
```

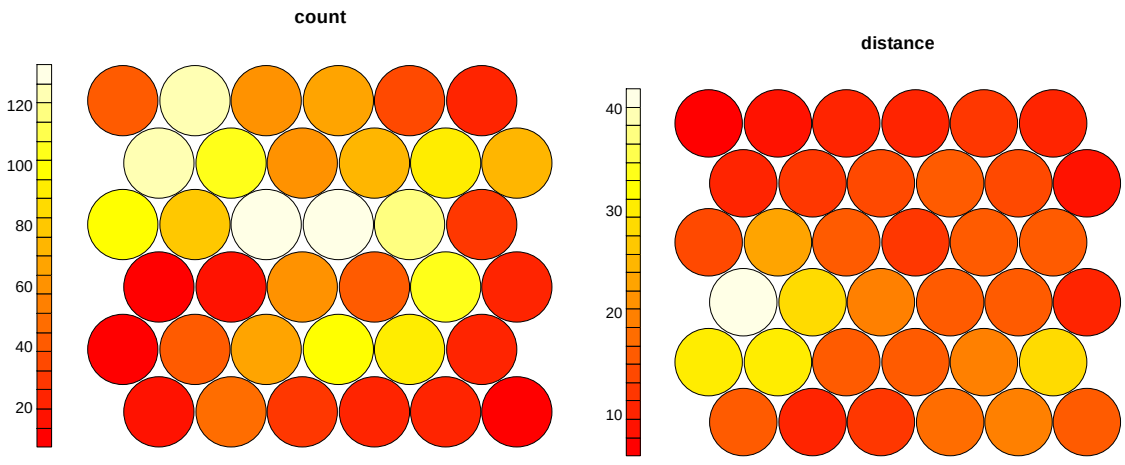


Figure 0-7 On the left: Lattice showing counts On the right: Lattice showing the U-matrix (distance)

Another very useful plot is the distance plot also referred to as the U-matrix. This plot is often used to further cluster the SOM lattice. You can create the distance plot as follows:

```
plot (remotemat.som, type="dist.neighbours", pch = 1, main="distance")
```

What we can do is use the U-matrix to further cluster the lattice. It has been proven that such a secondary clustering greatly improves the results. Manual clustering based on the U-matrix is often applied to much larger size lattices. It is more difficult on this small lattice. This is why we use a simple hierarchical clustering. Add the following lines to you script:

```
remotemat.hc <- cutree(hclust(dist(remotemat.som$codes)), 4)
add.cluster.boundaries(remotemat.som, remotemat.hc)
```

What we did is we clustered based on the distance between the neurons. This is what we visualized with our U-matrix. We applied hierarchical clustering (hclust) and added the boundaries to our previous plot.

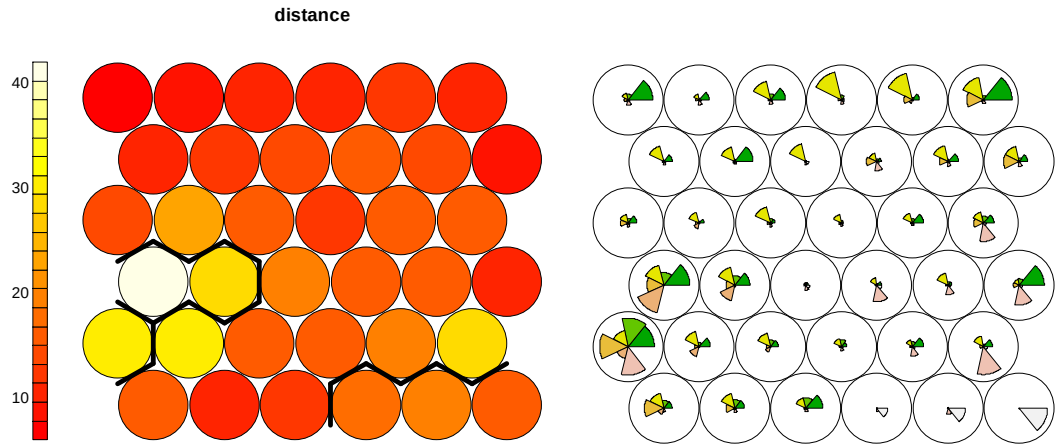


Figure 0-8U-matrix with cluster lines

What you see when you examine the outcome is that the cluster lines are positioned where the distances are the largest (the neurons are most dissimilar). Feel free to experiment with a different number of clusters and see how they are generated.

3. Bringing your SOM results back to a GIS

Eventually we would like to bring our clusters back to our GIS. What we would like to do is generate a file with the numbers of the neuron each of our records was mapped to.

```
#save the mapping to a file
```

```
write.csv(mapping, file="outputfile.csv")
```

When you open the file (for example in Excel) you see the following:

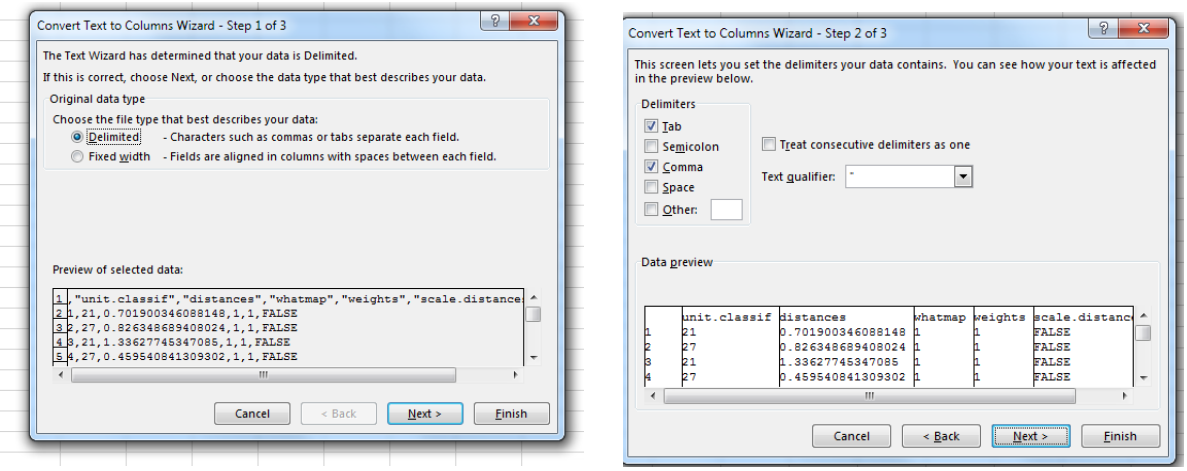
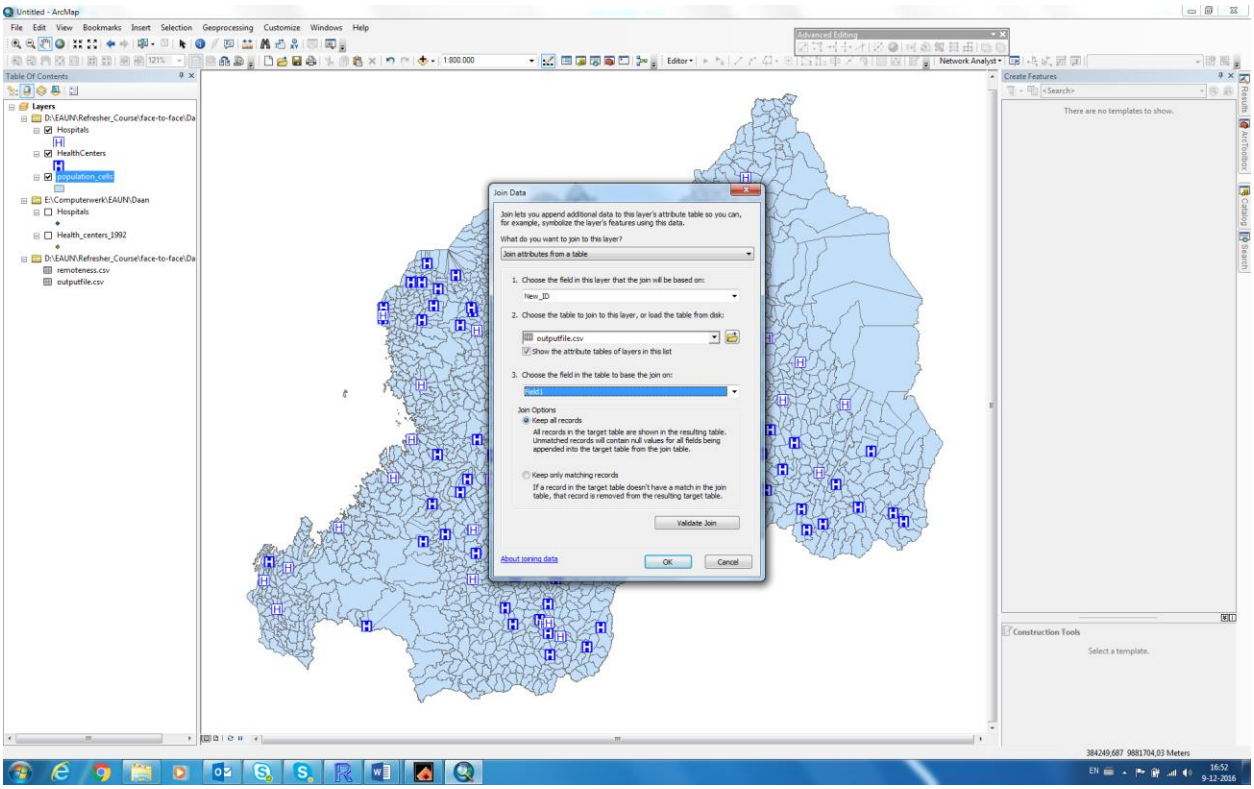


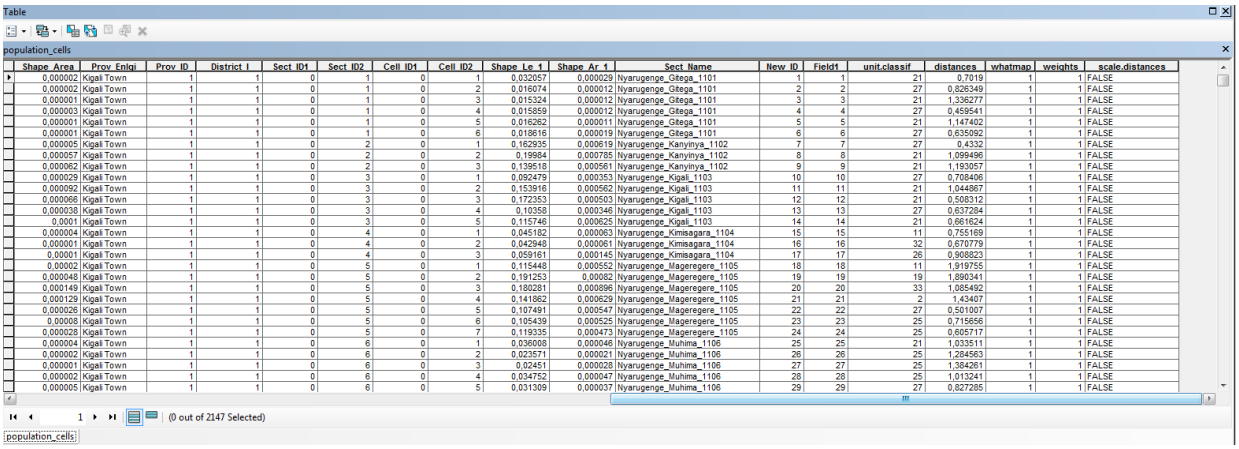
Figure 0-9 splitting the columns

All data is importorted into the same column so we need to split the data. Follow the dialog as you see above. In order to make the output spatial, we need to move the data back into a GIS.

Add the provided Shapefiles to a GIS. When you examine the Shapefile with the population cells you will see that it contains an FID (feature id) but that this field starts with the number 0 instead of 1. When you examine your output you see that the numbering starts with 1. This is why I already added a New_id field that can be used for joining the output file to the Shapefile.



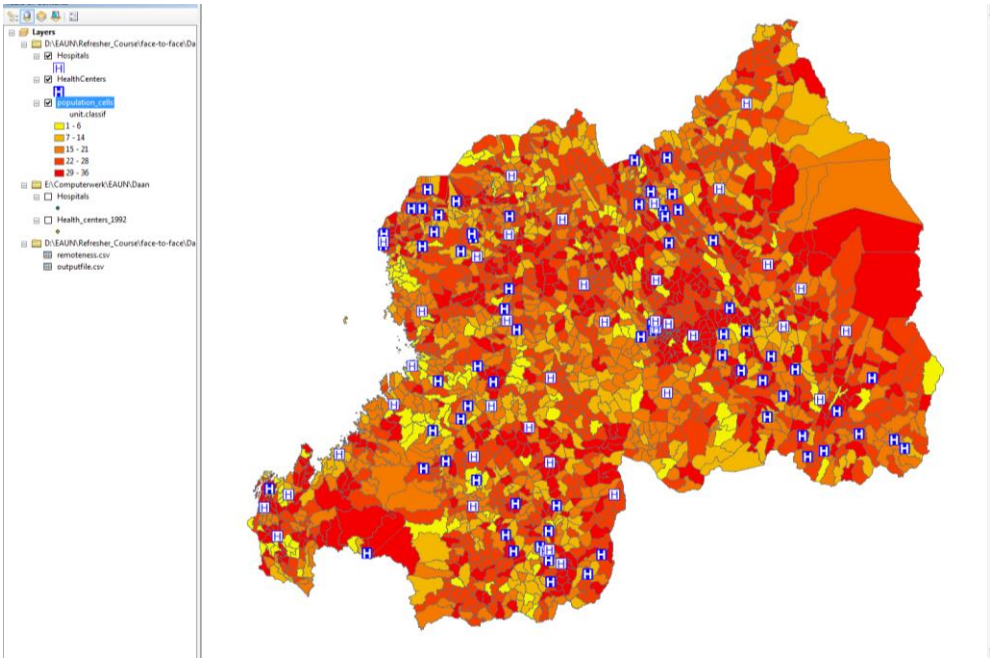
Use the id field to join your output file to the provided Shapefile and view the output attribute table to check if the output data has been added properly.



Shape_Area	Prov_Enlgi	Prov_ID	District_I	Sect_ID1	Sect_ID2	Cell_ID1	Cell_ID2	Shape_Le_1	Shape_Ar_1	Sect_Name	New_ID	Field1	unit.classif	distances	whatmap	weights	scale.distances
0,000002	Kigali Town	1	1	0	1	0	1	0,032057	0,000029	Nyarugenge_Gitega_1101	1	1	21	0,7019	1	1	FALSE
0,000002	Kigali Town	1	1	0	1	0	2	0,016074	0,000012	Nyarugenge_Gitega_1101	2	2	27	0,826349	1	1	FALSE
0,000001	Kigali Town	1	1	0	1	0	3	0,015324	0,000012	Nyarugenge_Gitega_1101	3	3	21	1,336277	1	1	FALSE
0,000003	Kigali Town	1	1	0	1	0	4	0,015859	0,000012	Nyarugenge_Gitega_1101	4	4	27	0,459541	1	1	FALSE
0,000001	Kigali Town	1	1	0	1	0	5	0,016262	0,000011	Nyarugenge_Gitega_1101	5	5	21	1,147402	1	1	FALSE
0,000001	Kigali Town	1	1	0	1	0	6	0,018616	0,000019	Nyarugenge_Gitega_1101	6	6	27	0,635092	1	1	FALSE
0,000005	Kigali Town	1	1	0	2	0	1	0,162935	0,000619	Nyarugenge_Kanyinya_1102	7	7	27	0,4332	1	1	FALSE
0,000057	Kigali Town	1	1	0	2	0	2	0,19884	0,000785	Nyarugenge_Kanyinya_1102	8	8	21	1,099496	1	1	FALSE
0,000062	Kigali Town	1	1	0	2	0	3	0,139518	0,000561	Nyarugenge_Kanyinya_1102	9	9	21	1,193057	1	1	FALSE
0,000029	Kigali Town	1	1	0	3	0	1	0,092479	0,000353	Nyarugenge_Kigali_1103	10	10	27	0,708406	1	1	FALSE
0,000092	Kigali Town	1	1	0	3	0	2	0,153916	0,000562	Nyarugenge_Kigali_1103	11	11	21	1,044867	1	1	FALSE
0,000066	Kigali Town	1	1	0	3	0	3	0,172353	0,000503	Nyarugenge_Kigali_1103	12	12	21	0,508312	1	1	FALSE
0,000038	Kigali Town	1	1	0	3	0	4	0,10358	0,000346	Nyarugenge_Kigali_1103	13	13	27	0,637284	1	1	FALSE
0,0001	Kigali Town	1	1	0	3	0	5	0,115746	0,000625	Nyarugenge_Kigali_1103	14	14	21	0,661624	1	1	FALSE
0,000004	Kigali Town	1	1	0	4	0	1	0,045182	0,000063	Nyarugenge_Kimisagara_1104	15	15	11	0,755169	1	1	FALSE
0,000001	Kigali Town	1	1	0	4	0	2	0,042948	0,000061	Nyarugenge_Kimisagara_1104	16	16	32	0,670779	1	1	FALSE
0,00001	Kigali Town	1	1	0	4	0	3	0,059161	0,000145	Nyarugenge_Kimisagara_1104	17	17	26	0,908823	1	1	FALSE
0,00002	Kigali Town	1	1	0	5	0	1	0,115448	0,000552	Nyarugenge_Mageregere_1105	18	18	11	1,919755	1	1	FALSE
0,000048	Kigali Town	1	1	0	5	0	2	0,191253	0,00082	Nyarugenge_Mageregere_1105	19	19	19	1,890341	1	1	FALSE
0,000149	Kigali Town	1	1	0	5	0	3	0,180281	0,000896	Nyarugenge_Mageregere_1105	20	20	33	1,085492	1	1	FALSE
0,000129	Kigali Town	1	1	0	5	0	4	0,141862	0,000629	Nyarugenge_Mageregere_1105	21	21	2	1,43407	1	1	FALSE
0,000026	Kigali Town	1	1	0	5	0	5	0,107491	0,000547	Nyarugenge_Mageregere_1105	22	22	27	0,501007	1	1	FALSE
0,00008	Kigali Town	1	1	0	5	0	6	0,105439	0,000525	Nyarugenge_Mageregere_1105	23	23	25	0,715656	1	1	FALSE
0,000028	Kigali Town	1	1	0	5	0	7	0,119335	0,000473	Nyarugenge_Mageregere_1105	24	24	25	0,605717	1	1	FALSE
0,000004	Kigali Town	1	1	0	6	0	1	0,036008	0,000046	Nyarugenge_Muhima_1106	25	25	21	1,033511	1	1	FALSE
0,000002	Kigali Town	1	1	0	6	0	2	0,023571	0,000021	Nyarugenge_Muhima_1106	26	26	25	1,284563	1	1	FALSE
0,000001	Kigali Town	1	1	0	6	0	3	0,02451	0,000028	Nyarugenge_Muhima_1106	27	27	25	1,384261	1	1	FALSE
0,000002	Kigali Town	1	1	0	6	0	4	0,034752	0,000047	Nyarugenge_Muhima_1106	28	28	25	1,013241	1	1	FALSE
0,000005	Kigali Town	1	1	0	6	0	5	0,031309	0,000037	Nyarugenge_Muhima_1106	29	29	27	0,827285	1	1	FALSE

Figure 0-10 joined table

Use the joined data to visualize. When I simply visualize using the unit.classif field I will not see much pattern.



But, when I manually apply the clustering as we generated earlier, you can see that some patterns can be revealed.

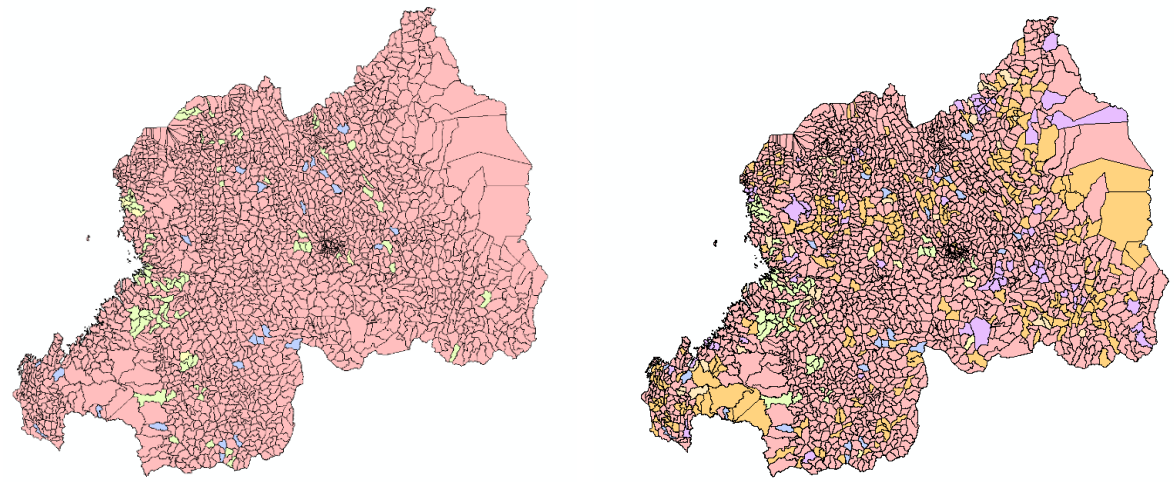


Figure 0-11 Left: with 4 classes and right with 7 classes

References:

1. Basara, H. and M. Yuan, *Community health assessment using self-organizing maps and geographic information systems.* International Journal of Health Geographics, 2008. **7**(1): p. 67.

2. Augustijn, E.-W. and R. Zurita-Milla, *Self-organizing maps as an approach to exploring spatiotemporal diffusion patterns.* International Journal of Health Geographics, 2013. **12**(1): p. 60.
3. Montagu, D. and G.M. Yamey, *Pay-for-performance and the Millenium Development Goals.* Lancet, 2011. **377**: p. 1383-11385.
4. Elridge, C. and N. Palmer, *Performance-based payment: some reflections on the discourse, evidence and unanswered questions.* Health policy and planning, 2009. **24**: p. 160-166.
5. Canavan, A., J. Toonen, and R. Elovainio, *Performance Based Financing: An international review of the literature.* KIT Development Policy & Practice, 2008.
6. Wehrens, R. and L.M.C. Buydens, *Self- and super-organizing maps in R: The kohonen package.* Journal of Statistical Software, 2007. **21**(5): p. 1-19.