Self-Organizing Maps (SOM)

In this lesson we'll learn the theory behind using Linear Self-Organizing Maps (SOM) as a clustering and mapping technique. We'll then use Self-Organizing Maps to cluster the UCI wine dataset in R.

Additional packages needed

To run the code you may need additional packages.

- If necessary install the followings packages.

```
install.packages("ggplot2");
install.packages("kohonen");
```

```
require(ggplot2)

## Loading required package: ggplot2

require(kohonen)

## Loading required package: kohonen

## Loading required package: class

## Loading required package: MASS
```

Data

We will be using the UCI Machine Learning Repository: Wine Data Set. These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.

The attributes are:
1) Alcohol
2) Malic acid
3) Ash
4) Alcalinity of ash
5) Magnesium
6) Total phenols
7) Flavanoids
8) Nonflavanoid phenols
9) Proanthocyanins
10) Color intensity
11) Hue

12) OD280/OD315 of diluted wines
13) Proline

Feel free to tweet questions to
[@NikBearBrown](https://twitter.com/NikBearBrown)

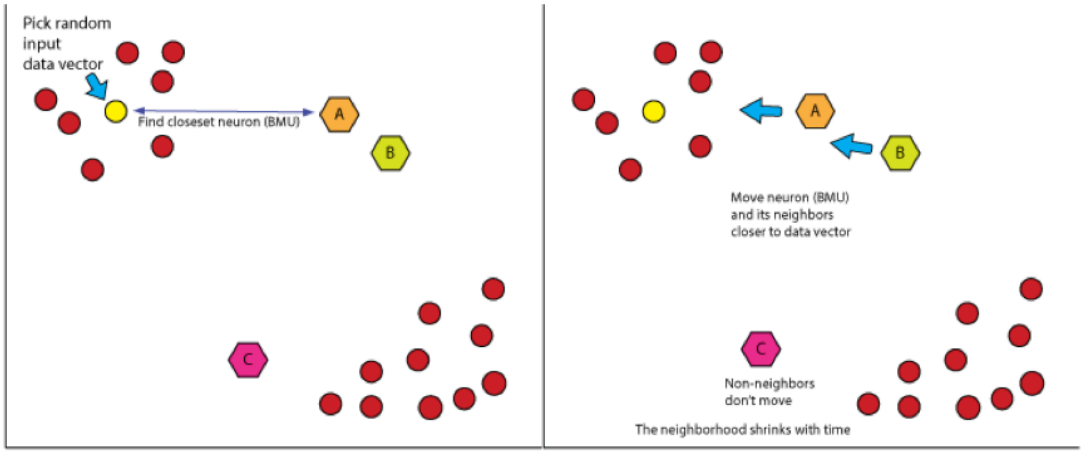
```
# Load our data
data(wines)
head(wines)
```

```
##      alcohol malic acid  ash ash alkalinity magnesium tot. phenols
## [1,]   13.20       1.78 2.14           11.2       100         2.65
## [2,]   13.16       2.36 2.67           18.6       101         2.80
## [3,]   14.37       1.95 2.50           16.8       113         3.85
## [4,]   13.24       2.59 2.87           21.0       118         2.80
## [5,]   14.20       1.76 2.45           15.2       112         3.27
## [6,]   14.39       1.87 2.45           14.6        96         2.50
##      flavonoids non-flav. phenols proanth col. int. col. hue OD
ratio
## [1,]       2.76              0.26    1.28      4.38     1.05
3.40
## [2,]       3.24              0.30    2.81      5.68     1.03
3.17
## [3,]       3.49              0.24    2.18      7.80     0.86
3.45
## [4,]       2.69              0.39    1.82      4.32     1.04
2.93
## [5,]       3.39              0.34    1.97      6.75     1.05
2.85
## [6,]       2.52              0.30    1.98      5.25     1.02
3.58
##      proline
## [1,]    1050
## [2,]    1185
## [3,]    1480
## [4,]     735
## [5,]    1450
## [6,]    1290
```

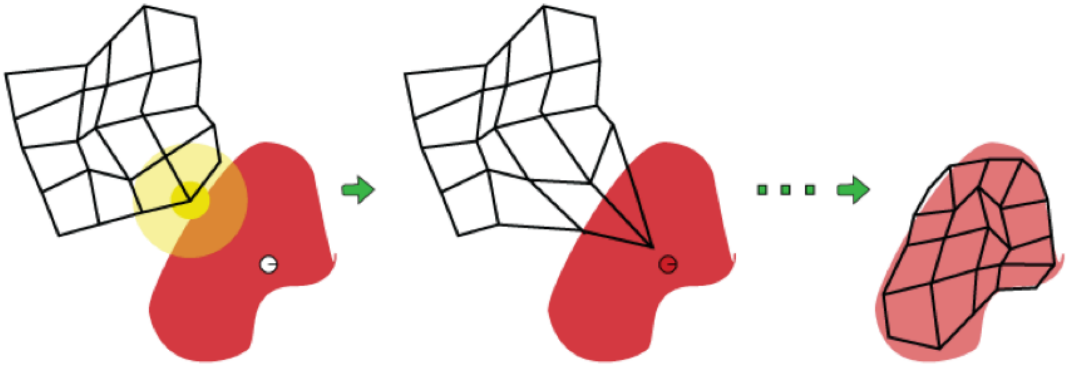
Self-Organizing Maps (SOM)

A self-organizing map (SOM) or self-organizing feature map (SOFM) is an artificial neural network (ANN) that is trained using unsupervised learning to produce a low-dimensional (typically two-dimensional), discretized representation of the input space of the training samples, called a map. This makes SOMs useful for visualizing low-dimensional views of high-dimensional data. In this sense, SOMs are similar to multidimensional scaling. Self-organizing maps with a small number of nodes

behave in a way that is similar to K-means, larger self-organizing maps rearrange data in a way that is fundamentally topological in character.

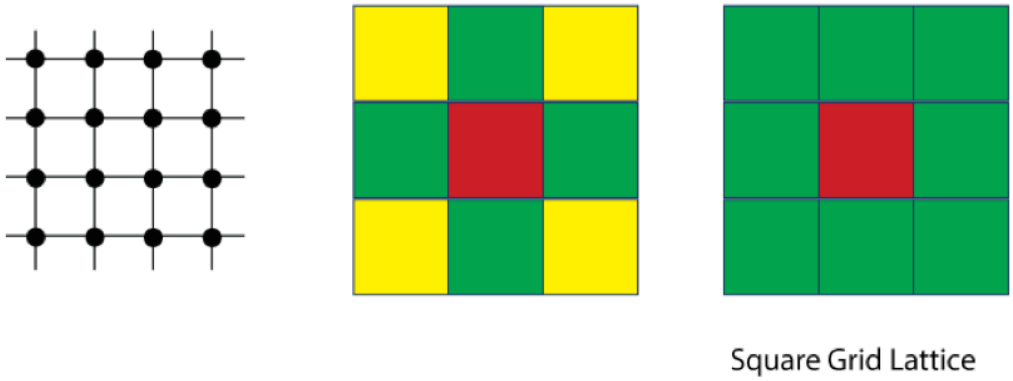


SOM Algorithm

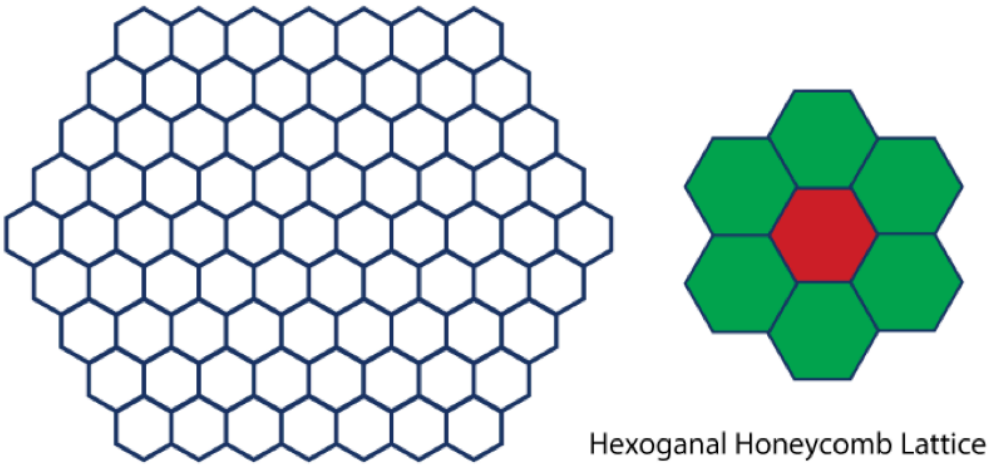


SOM Training

SOM neighborhood topology



Square Grid Lattice SOM]



Hexoganal Honeycomb Lattice SOM]

The grids have an x dimension and y dimension. The topology of the grid is rectangular or hexagonal.

SOM neighborhood functions

While the neighborhood function $\theta(u, v, s)$ usually depends on the lattice distance between the BMU (neuron u) and neuron v. Other functions such as a Gaussian function or a radius distance centered around the neuron are also common choices.

SOM Algorithm

A. Randomize the map's nodes' weight vectors. The weights of the neurons are initialized either to small random values or sampled evenly from the subspace spanned by the two largest principal component eigenvectors.

B. Grab an input vector $\mathbf{D}(\mathbf{t})$. t is the index of the target input data vector in the input data set $\mathbf{D}$ Each data point, $\mathbf{D}(\mathbf{t})$, needs to be mapped to a "neuron."

C. Traverse each node in the map. Use the Euclidean distance or another similarity metric between the input vector $\mathbf{D}(\mathbf{t})$ and the map's node's weight vector. The node that produces the smallest distance (i.e. the best matching unit, BMU) is associated with that neuron.

D. Update the nodes in the neighborhood of the BMU (including the BMU itself) by pulling them closer to the input vector. This is analogous to updating the new means to be the centroids of the observations in the new clusters the means in k-means clustering. That is,

$$Wv(s+1) = Wv(s) + \theta(u, v, s)\alpha(s)(D(t) - Wv(s))$$

Where

- s is the current iteration
- λ is the iteration limit. Sometimes this is called the maximum number of 'epochs'
- v is the index of the node in the map
- $\mathbf{W_v}$ is the current weight vector of node v
- u is the index of the best matching unit (BMU) in the map
- $\Theta(u, v, s)$ is a restraint due to distance from BMU, usually called the neighborhood function,
- $\alpha(s)$ is a learning restraint due to iteration progress. Sometimes this is called the learning rate.

E Increase s and repeat from step B while $s < \lambda$

Note that the neighborhood function $\Theta(u, v, s)$ depends on the lattice distance between the BMU (neuron u) and neuron v. In the simplest form it is 1 for all neurons close enough to BMU and 0 for others, but a Gaussian function is a common choice, too. Regardless of the functional form, the neighborhood function shrinks with time.

A variant algorithm:

A. Randomize the map's nodes' weight vectors
B. Traverse each input vector in the input data set
C. Traverse each node in the map
D. Use the Euclidean distance formula to find the similarity between the input vector and the map's node's weight vector E. Track the node that produces the smallest distance (this node is the best matching unit, BMU) Update the nodes in the neighborhood of the BMU (including the BMU itself) by pulling them closer to the input vector $Wv(s+1) = Wv(s) + \theta(u, v, s)\alpha(s)(D(t) - Wv(s))$ F. Increase s and repeat from step B while $s < \lambda$

The kind of training is called *competitive learning* n which nodes compete for the right to respond to a subset of the input data.

SOM Usage

Using Self-Organising Maps are typcially done as follows:

Select the size and type of the map. The shape can be hexagonal or square. Note that hexagonal grid has six immediate neighbors whereas a square usually has four. This topology determines the number of "neurons."

The algorithm then

A. Initializes all node weight vectors.
B. Chooses a random data point from the training data to present to the SOM.
C. Finds the "Best Matching Unit" (BMU) in the map. Determine the nodes within the "neighborhood" of the BMU.
D. The size of the neighborhood decreases with each iteration.
E. Adjust weights of nodes (neurons) in the BMU neighborhood towards a chosen datapoint. - The learning rate decreases with each iteration.
- The magnitude of the adjustment is proportional to the proximity of the node to the BMU.

Repeat Steps B-E for a fixed number of iterations or until convergence.

SOM Pros and Cons

Pros:

- Intuitive method.
- Simple algorithm.
- New data points can be mapped to trained model for predictive purposes.

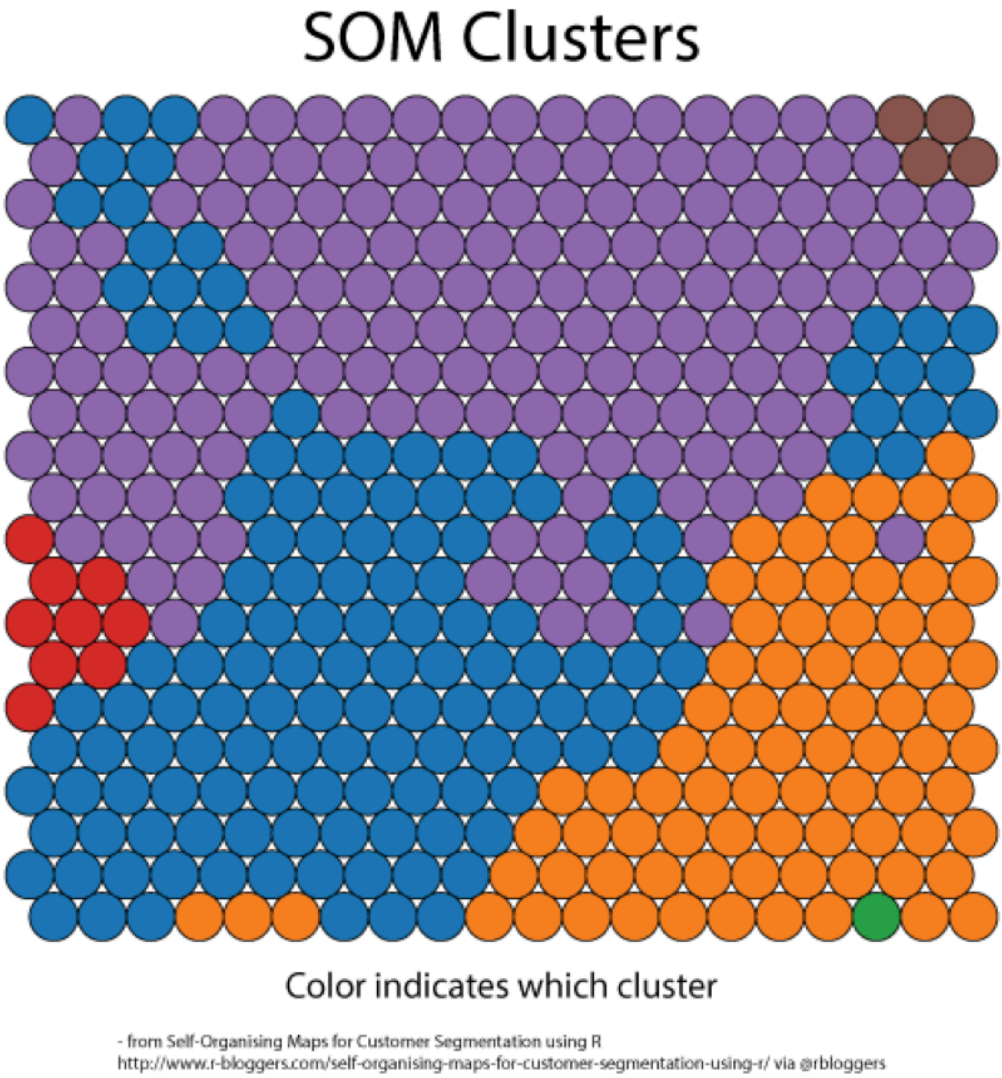
Cons:

- Requires clean, numeric data.

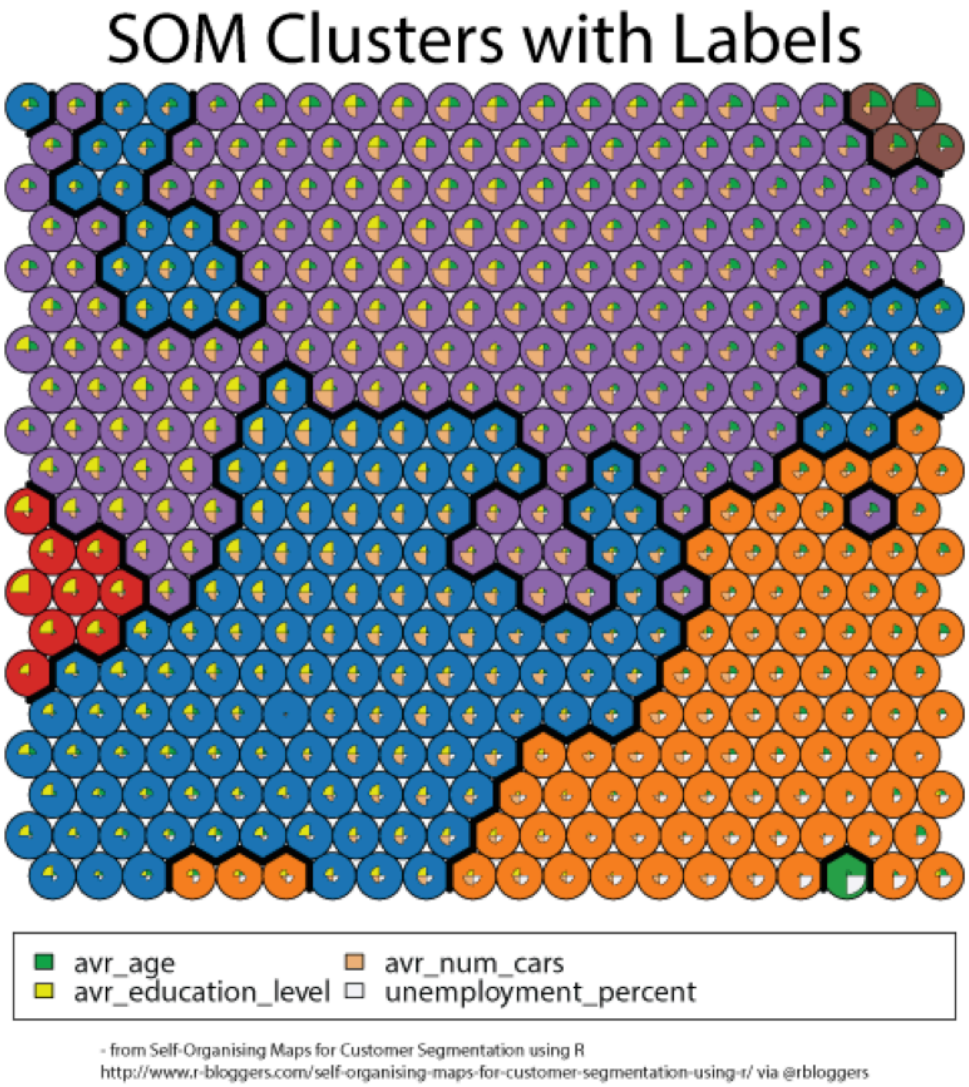
SOM Visualization

The Self-Organizing Maps projects high-dimensional data onto a two-dimensional map. The projection preserves the topology of the data so that similar data items will be mapped to nearby locations on the map. As such one of its great strengths is high-dimensional data visualization, akin to multidimensional scaling (MDS).

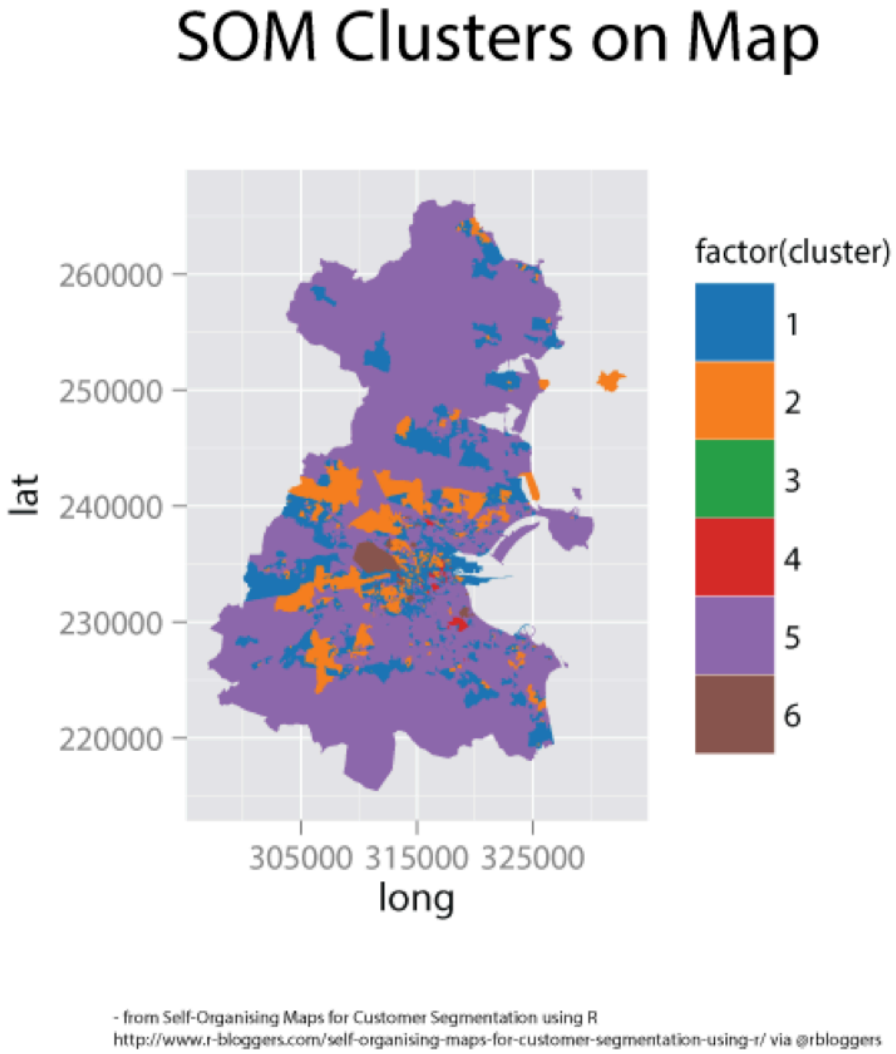
SOM Cluster Maps



SOM Clusters
- from Self-Organising Maps for Customer Segmentation using R via @rbloggers

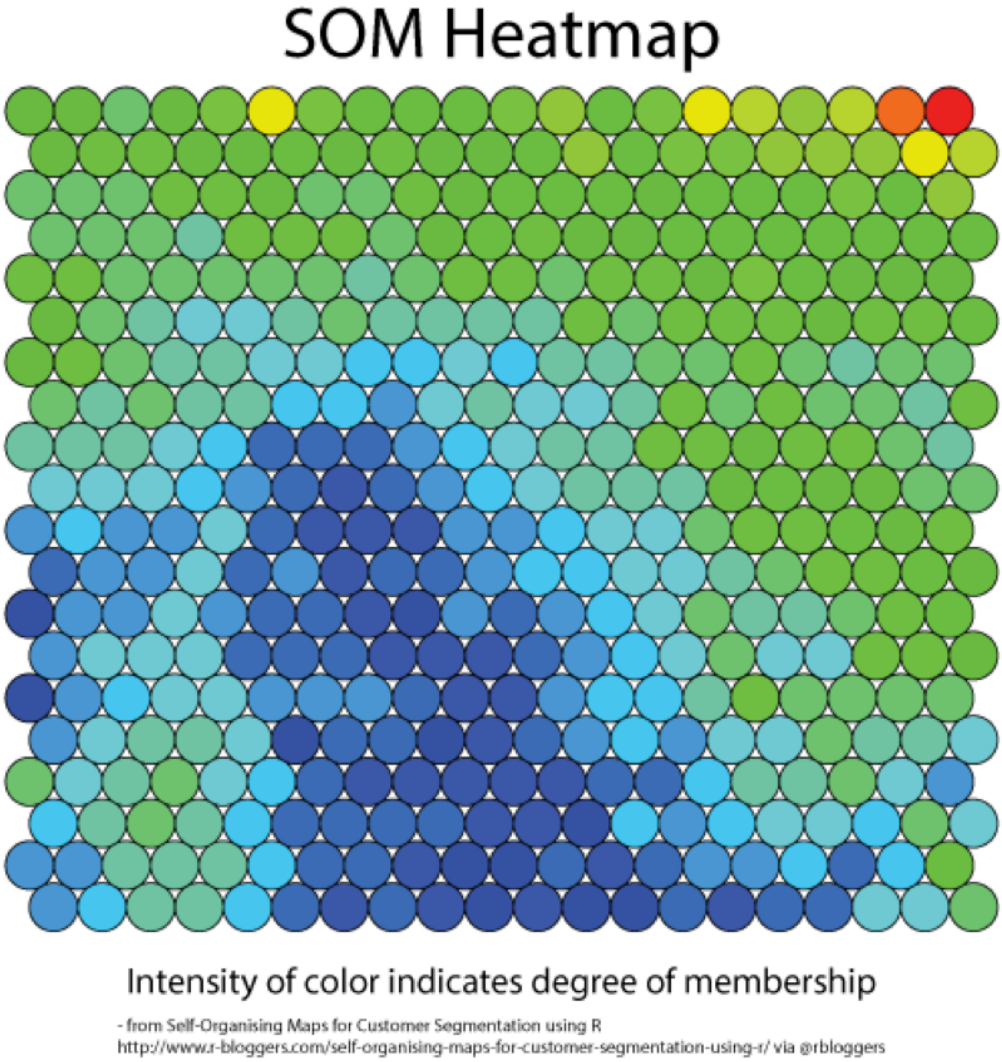


SOM Clusters with Labels
- from Self-Organising Maps for Customer Segmentation using R via @rbloggers



SOM Clusters on Map
- from Self-Organising Maps for Customer Segmentation using R via @rbloggers ## SOM Heatmaps

A heat map is a graphical representation of data where a gradient of values are represented as colors.



SOM Heatmap
- from Self-Organising Maps for Customer Segmentation using R via @rbloggers

SOMs in R

Clustering the data with a 5x5 hexagonal grid.

```
training <- sample(nrow(wines), 120)
Xtraining <- scale(wines[training, ])
# 5, 5, "hexagonal"
fit.som <- som(Xtraining, somgrid(5, 5, "hexagonal"))
map.som <- map(fit.som,
               scale(wines[-training, ],
                     center=attr(Xtraining, "scaled:center"),
                     scale=attr(Xtraining, "scaled:scale")))
fit.som
```

```
## som map of size 5x5 with a hexagonal topology.
## Training data included.

map.som

## $unit.classif
##  [1]  1  1 11  1  2  3  7  2  7 16 17 16  6  7  1  7  2 23  8 19 23
15 19
## [24] 23 23 19 23 18 25 23 19 19 24 15  5 14 14  4 14  4 14 10  5  5
5 10
## [47] 10  5 10  4  4  4  4 10  5  5  5
##
## $distances
##  [1]  0.8110809  4.4274322  1.7157470  4.5153648  2.3955237
6.9131644
##  [7]  3.6446892  2.7956718  3.0688226  1.9255938  6.7051503
4.1329453
## [13]  4.7241695  2.1253630  2.9199586  2.7634707  1.7619999
5.8280694
## [19]  8.0207076  6.5523402  4.6175478  7.1683701  7.2544121
3.0765295
## [25]  2.3257569 13.1411962  1.8081202  3.3206756  9.7874075
1.9958500
## [31]  3.2526887  1.0339188  8.4492901  6.0071278  6.0648093
8.6602095
## [37]  4.0119393  6.0038566  6.2360543  8.2503782  3.8240620
4.5128385
## [43]  8.1457417  0.8847921  4.3701976  8.2327580  9.5001545
5.2799640
## [49]  8.6797738  8.5980421  3.3706161  2.8423445  9.2926421
3.1776709
## [55]  5.0264030  3.8409749  3.1407094
##
## $whatmap
## [1] 1
##
## $weights
## [1] 1
##
## $scale.distances
## [1] FALSE
```

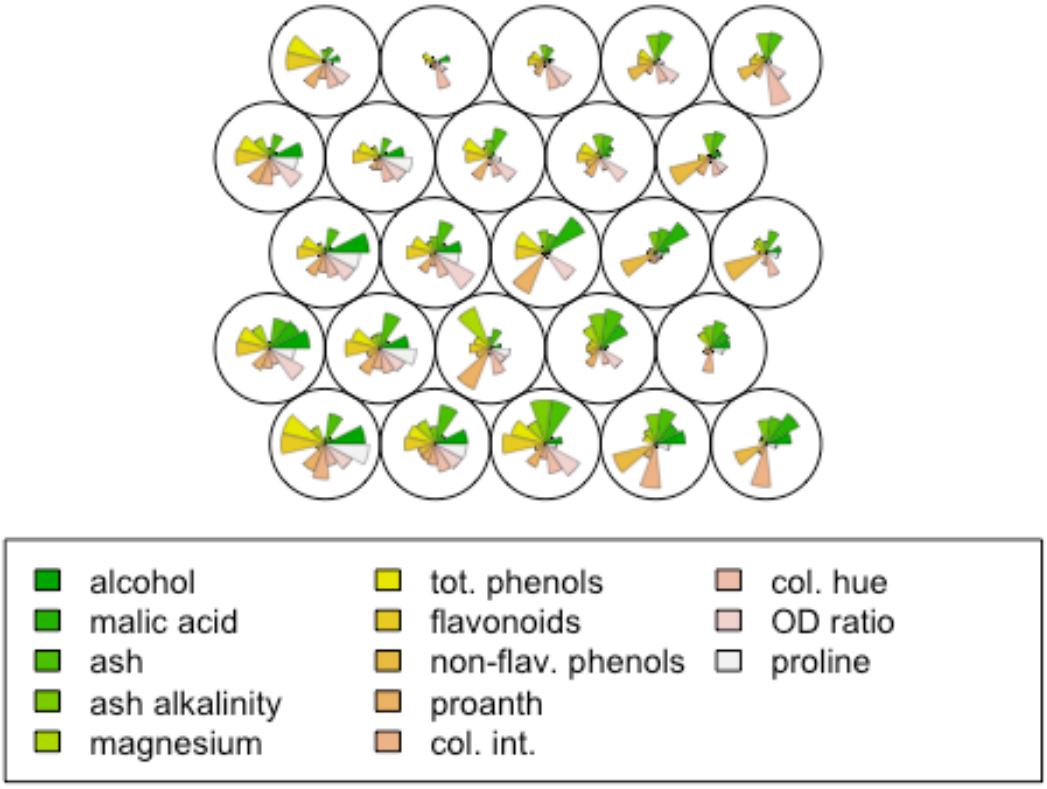
SOM Visualization in R

Code Plot

The codebook vectors are visualized in a segments plot, which is the default plotting type. The the code plot shows each cluster and the node weight vectors or "codes" associated with each node. That is, the fan chart in the center of the clusters reveals the characteristics that define how much of each attribute were clustered into each

particular cluster. High alcohol levels, for example, are associated with wine samples projected in the bottom right corner of the map, while color intensity is largest in the bottom left corner.

```
plot(fit.som,"codes")
```



Training Plot ("changes")

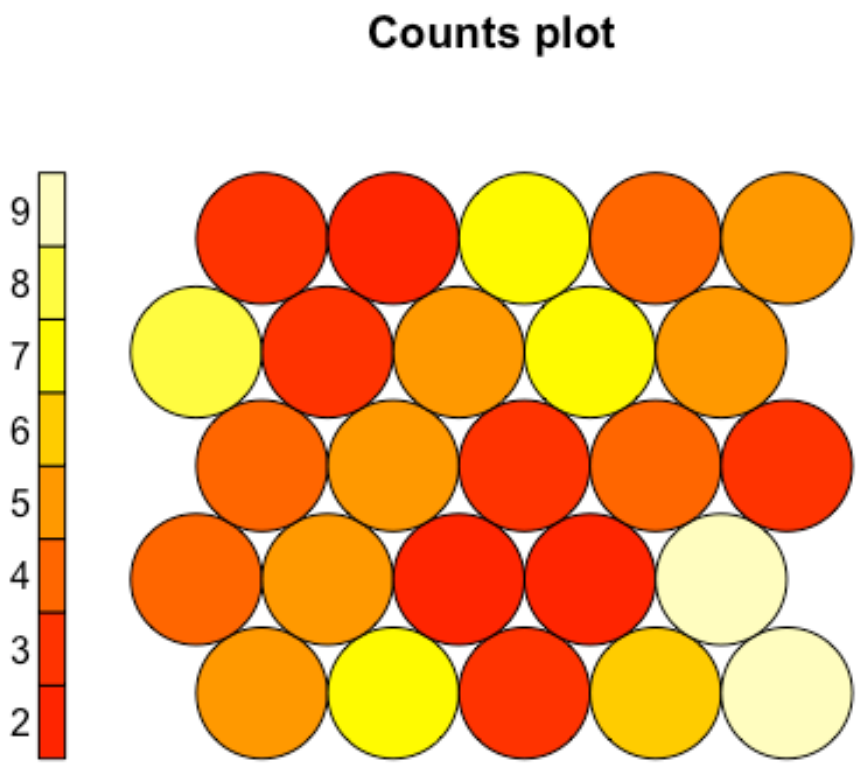
This shows a hundred iterations. This shows the mean distance to the closest codebook vector during training. Training decreases the mean distance with iterations and typically plateaus at around 40 iterations.

```
plot(fit.som,"changes")
```

Count Plot ("counts")

This tells you how many items are in each of the clusters. The count plot can be used as a quality check. Ideally you want a uniform distribution. That is, you would like all the data not to be bunched up in a few nodes (i.e. neurons)

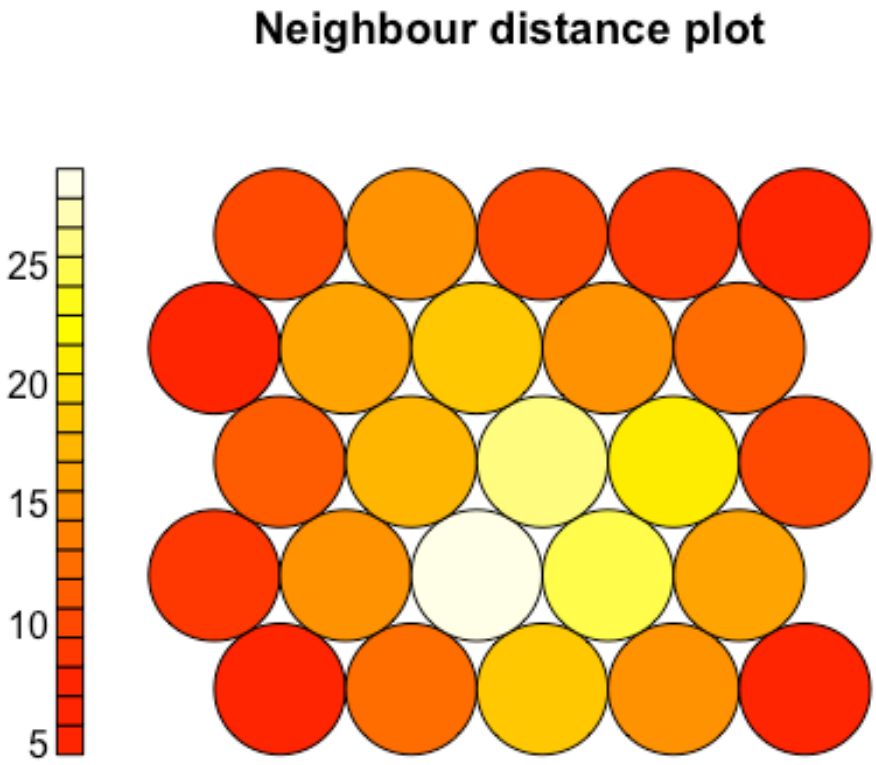
```
plot(fit.som,"counts")
```



Distance Neighbour Plot ("dist.neighbours")

This is sometimes called the "U-Matrix." The U-Matrix value of a particular node is the average distance between the node's weight vector and that of its closest neighbors. Areas of low neighbor distance indicate groups of nodes that are similar and the further apart nodes indicate natural "borders" in the map. That is, units near a class boundary can be expected to have higher average distances to their neighbors.

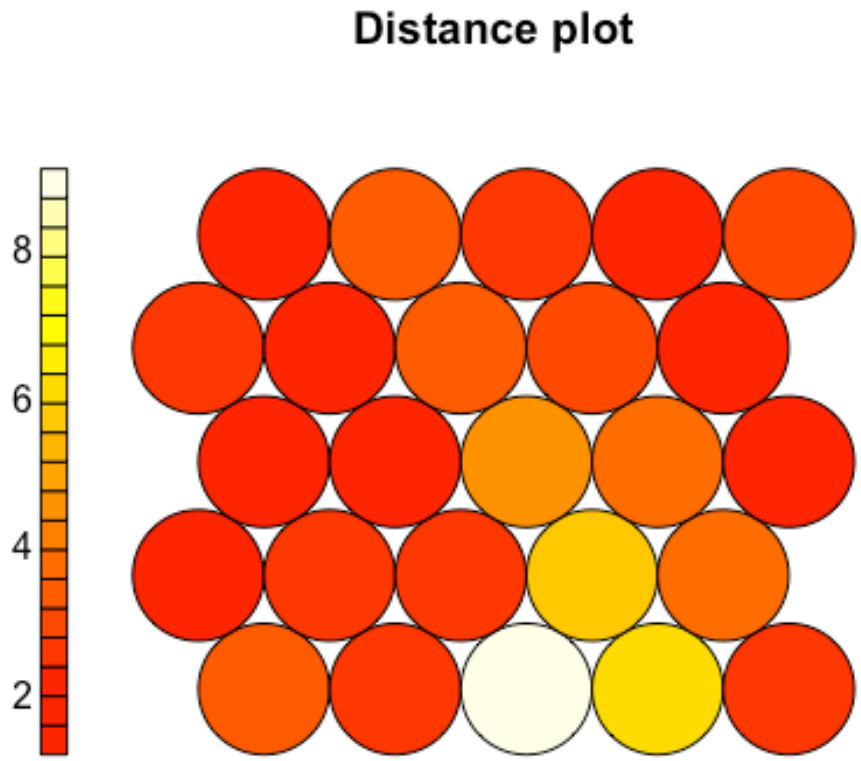
```
plot(fit.som,"dist.neighbours")
```



Quality Plot ("quality")

This 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.

```
plot(fit.som,"quality")
```



Clustering the data with a 3x3 rectangular grid.

Clustering the data with a 3x3 rectangular grid.

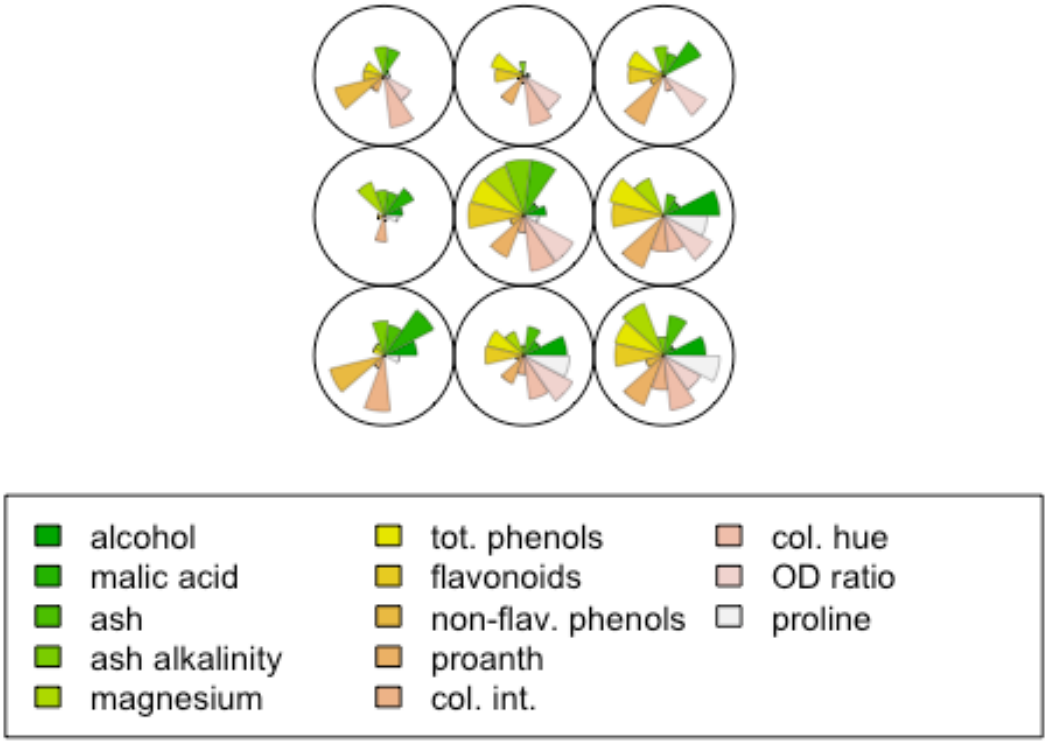
```
fit.som <- som(Xtraining, somgrid(3, 3, "rectangular"))
map.som <- map(fit.som,
               scale(wines[-training, ],
                     center=attr(Xtraining, "scaled:center"),
                     scale=attr(Xtraining, "scaled:scale")))
fit.som

## som map of size 3x3 with a rectangular topology.
## Training data included.

map.som

## $unit.classif
##  [1] 6 6 2 6 3 5 3 2 2 6 2 6 9 3 6 2 3 8 8 7 8 7 9 8 8 9 8 9 7 8 7 9
7 1 1
## [36] 1 1 1 1 1 1 4 1 1 1 4 4 1 1 1 1 1 1 4 1 1 1
##
## $distances
##  [1]  3.432411  4.823292  2.508460  8.154290  1.616751  5.633223
3.457345
```

```
##  [8]  4.555783  2.422346  2.200003  5.601246  3.052968  5.734533
2.696659
## [15]  3.639929  4.148808  1.473093  6.512619 23.197642  7.926846
5.861945
## [22]  6.932364  5.678206  2.598461  1.268804 12.117355  2.683129
7.021845
## [29]  7.857325  3.386405  5.155583  2.957746  9.375959 10.426583
5.206099
## [36]  8.223480  4.089996  6.062301  5.937515  7.214893  6.489321
4.575956
## [43] 10.855920  1.078377  5.523168  9.565401 10.238888  5.189005
8.112427
## [50] 12.437593  4.331263  4.438010  9.346687  3.078061  6.101775
3.820527
## [57]  4.108162
##
## $whatmap
## [1] 1
##
## $weights
## [1] 1
##
## $scale.distances
## [1] FALSE
```

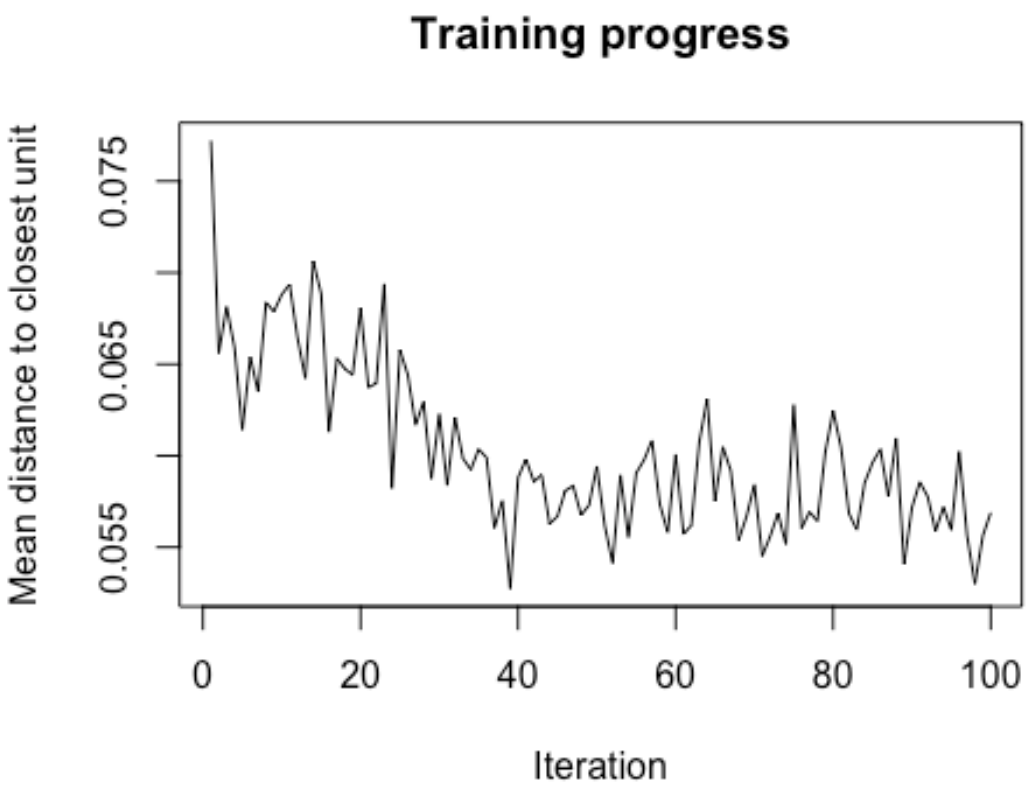
SOM visualization 3x3 rectangular grid

Plotting the 3x3 rectangular grid fit.

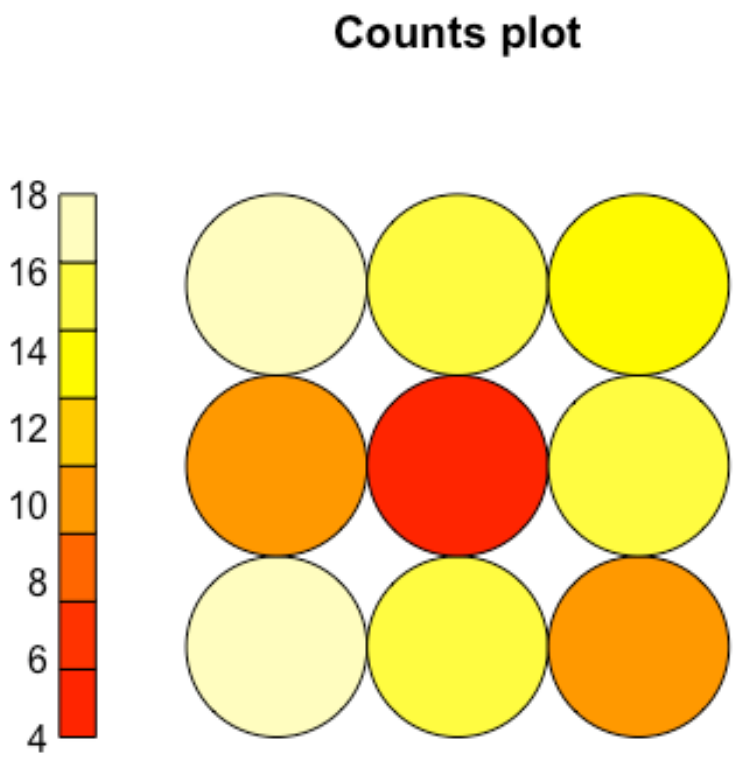
```
plot(fit.som)
plot(fit.som,"codes")
```



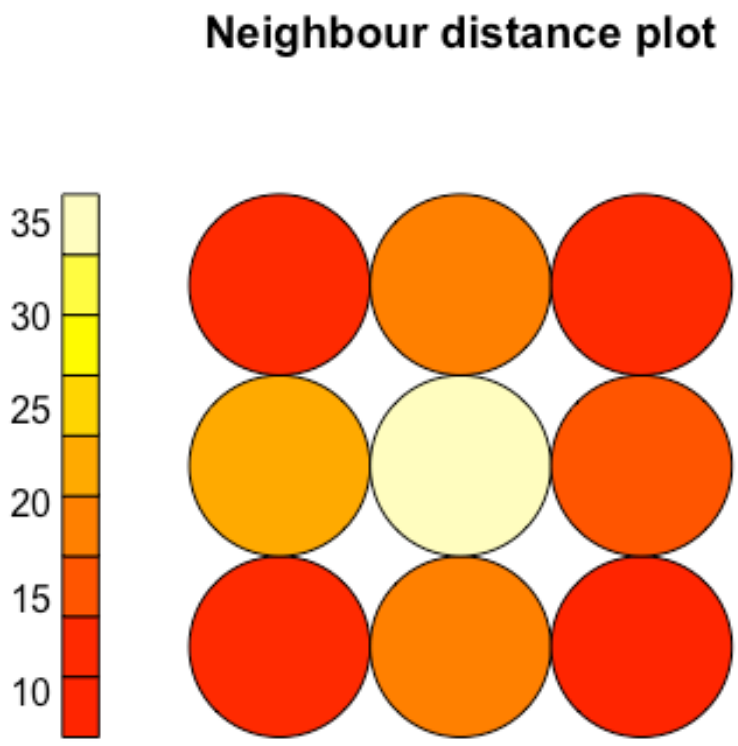

```
plot(fit.som,"changes")
```



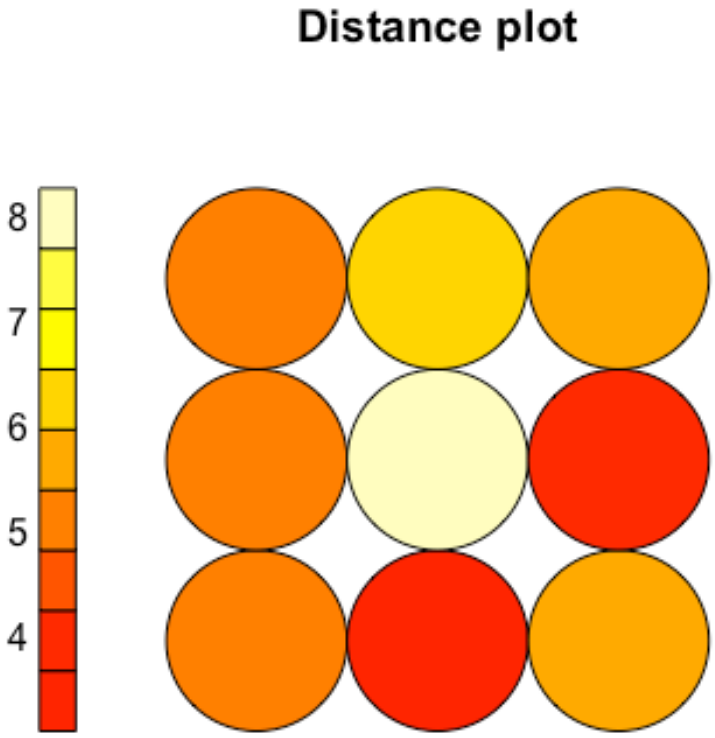

```
plot(fit.som,"counts")
```




```
plot(fit.som,"dist.neighbours")
```




```
plot(fit.som,"quality")
```



Resources

- Self-Organising Maps for Customer Segmentation using R via @rbloggers
- Self-Organizing Maps (SOM)
- On the creation of an extended Self Organizing Map program in R