Lecture 9. Self-Organizing Map (SOM)

Unsupervised learning with self-organization

Zhonghai Lu

KTH Royal Institute of Technology

May 6, 2024

Table of contents

What is SOM?

- Self-Organizing Map (SOM) is one of the most popular neural network models, but a different type. It is also named ”Kohonen model/map/network”.
- It is based on unsupervised learning, which means no labeled data, unknown characteristics of the input data.
  - For example, use SOM for clustering data without knowing the class memberships of the input data.
- It belongs to the category of *competitive learning* (vs. *error-correction learning*, i.e. error back-propagation with gradient descent) networks.
- SOM can be used to detect features inherent to the problem and thus has also been called SOFM, Self-Organizing Feature Map.

Why is SOM special?

- It provides a topology preserving mapping from the high dimensional space to map units or neurons, which usually form a two-dimensional lattice. This means mapping from high dimensional space onto a 2D plane.
  - The property of topology preserving means that the mapping preserves the relative distance between the data points.
  - Data points near each other in the input space are mapped to nearby neurons in the SOM. SOM can thus serve as a cluster analyzing tool of high-dimensional data.
- SOM has the capability to generalize.
  - The network can recognize or characterize inputs it has never encountered before.
  - A new input is assimilated with the neuron to which it is mapped.
- It is different from the neural networks we have learnt so far. How?

SOM

- SOM is a two-layer neural network: input layer and output layer
  - For the input layer, the number of neurons is determined by the dimension of the input vector.
  - For the output layer, the neurons have connections to neighbor neurons, indicating only adjacency (neighborhood) of nodes.
    The number of output neurons is a design choice.
  - The two layers are fully connected, meaning a large amount of weights.

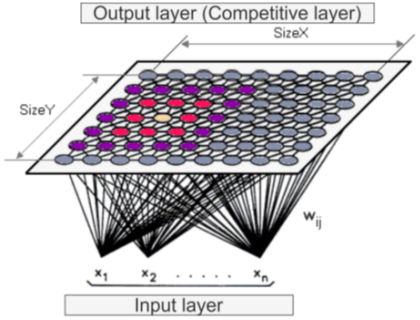


Figure: SOM structure

SOM: 1D output layer

- 1D output layer gives us a clearer picture on the input layer and the weight vectors of the output layer.

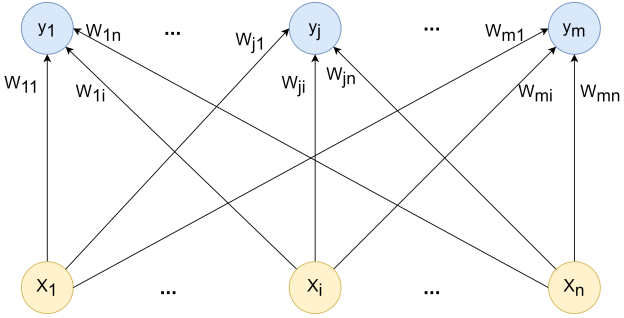


Figure: SOM with one-dimensional output layer

SOM

- Typically the output layer is a 2D array of neurons "connected" to each other via a rectangular or hexagonal topology. The topological relations are shown by lines between the neurons.

$$M = m_1, \cdots, m_{pxq}$$

- One neuron is a weight vector (some times called a codebook vector), which has the same dimension as the input vectors (n-dimensional).

$$m_i = [m_{i1}, \cdots, m_{in}]$$

- The neurons are connected to adjacent neurons by a neighborhood relation. This dictates the topology or structure of the map.

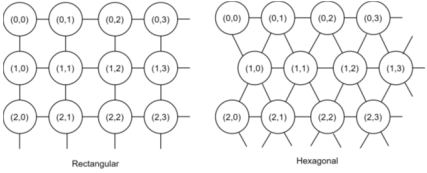
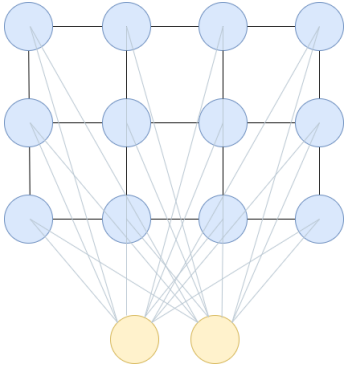


Figure: Typical output-layer topologies of SOM

A SOM example

- A 2D lattice of 3×4 'nodes', each of which is fully connected to the input layer representing a two-dimensional vector.
- Each node has a specific topological position (an (x, y) coordinate) and contains a vector of weights of the same dimension as the input vectors.
  - If the training data consists of vectors, V, of n dimensions: $V_1, V_2, \cdots, V_n$, Then each node contains a corresponding weight vector W, of n dimensions: $W_1, W_2, \cdots, W_n$
- The lines connecting the nodes represent adjacency and do not signify a connection as normally indicated for a neural network.



Topology relation and neighborhood

- One can also define a distance between the map units according to their topology relations.
- Adjacent neighbors, N_c, belong to the neighborhood of the neuron, M_c.
  The neighborhood function should be a decreasing function of time: $N_c = N_c(t)$.
- Neighborhoods of different sizes are illustrated in e.g. a hexagonal lattice. In the smallest hexagon, there are all the neighbors belonging to the smallest neighborhood of the neuron.

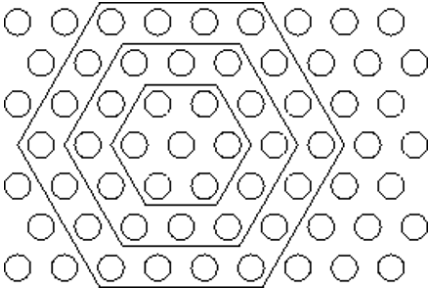


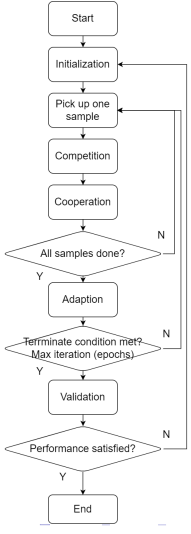
Figure: Neighborhood of a given winner unit

Training

- Training is an iterative process through time. It requires a lot of computational effort and thus is time-consuming.
- The training consists of drawing sample vectors from the input data set and ”teaching” them to the SOM.
- The teaching consists of choosing a winner unit, called best matching unit (BMU) by a similarity measure and updating the values of weight vectors in the neighborhood of the BMU.
- This process is repeated a number of times.

Network training

- Algorithmic steps:
  - Start: determine a SOM network X $\times$ Y
  - Initiation: initialize the network weights
  - Competition: find BMU for a given sample
  - Cooperation: update weights in the neighbor region of BMU
  - Adaptation: adjust learning rate and neighborhood size
  - Validation: test if the SOM performs well for new data
- Three loops:
  - 1st loop: for all input vectors
  - 2nd loop: for all iterations.
    - One iteration is one epoch which uses all input vectors once.
    - For each epoch, update learning rate, neighborhood size.
  - 3rd loop: Repeat from initialization if generalization performance is not satisfactory.



Initialization

Kohonen presents three different types of network initializations in his book [Teuvo Kohonen, 1995]:

- Random initialization
- Initialization using initial samples
- Linear initialization.

Initialization

- Random initialization means simply that random values are assigned to weight vectors. This is the case if nothing or little is known about the input data at the time of the initialization.
- Initial samples of the input data set can be used for weight vector initialization. This has the advantage that the points automatically lie in the same part of the input space with the data.
- One initialization method utilizes the principal component (PCA) analysis of the input data. The weight vectors are initialized to lie in the same input space that is spanned by two eigenvectors corresponding to the largest eigenvalues of the input data. This has the effect of stretching the SOM to the same orientation as the data having the most significant amounts of energy.

Competition

- In one training step, one sample vector is drawn randomly from the input data set.
- This vector is fed to all units in the network and a similarity measure is calculated between the input data sample and all the weight vectors.
- The best-matching unit (BMU) is chosen to be the weight vector with greatest similarity with the input sample.
- The similarity is usually defined by means of a distance measure. For example in the case of Euclidean distance, the BMU is the closest neuron to the sample in the input space.

Euclidean distance

- The Euclidean norm of vector x is defined as

$$||x|| = \sqrt{\sum_{i=1}^{n} x_i^2}$$

- The Euclidean distance is defined in terms of the Euclidean norm of the difference between two vectors:

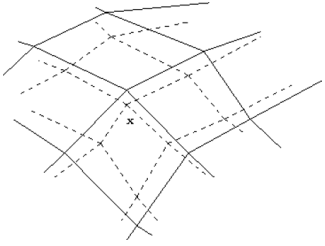
$$d_E(x, y) = ||x - y||$$

- The BMU, usually noted as m_c, is the weight vector that matches a given input vector x best. It is defined formally as the neuron for which

$$||x - m_c|| = \min_i \{||x - m_i||\}$$

SOM updating

- After finding the BMU, units in the SOM are updated. During the update procedure, the BMU is updated to be a little closer to the sample vector in the input space.
- The topological neighbors of the BMU are also similarly updated. This update procedure stretches the BMU and its topological neighbors towards the sample vector.
- The computational effort consists of finding a BMU among all the neurons and updating the weight vectors in the neighborhood of the winner unit.
  - If the neighborhood is large, there are a lot of weight vectors to be updated. This is the case in the beginning of the training process, where it is recommended to use large neighborhoods.
  - In the case of large networks, relatively larger portion of the time is spent looking for a winner neuron.



Updating effect

- Each update can be thought of as a shift in the topology of a neighborhood local to the output point.
- With sufficiently many iterations through this process, the nodes will form the dimension-reduced feature space over the original input which best preserves local topology:

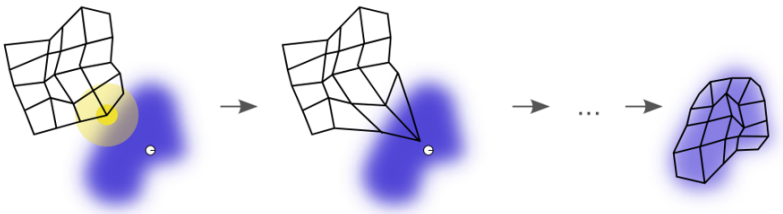


Figure: Effect of updating weight vectors.

Source: https://www.kaggle.com/residentmario/self-organizing-maps

Weight-updating rule

- The SOM update rule or formula for any unit m_i:

$$m_i(t+1) = m_i(t) + h_{ci}(t) \cdot [x(t) - m_i(t)]$$

  where t denotes time, since this is a training process through time.
- The $x(t)$ is the input vector drawn from the input data set at time t.
- $h_{ci}(t)$ is a non-increasing neighborhood function around the winner unit m_c.
  - When $h_{ci}(t) = 1$, $m_i(t+1) = m_i(t) + 1 \cdot [x(t) - m_i(t)] = x(t)$. Update to be the input vector.
  - When $h_{ci}(t) = 0$, $m_i(t+1) = m_i(t)$. No update.

Neighborhood function

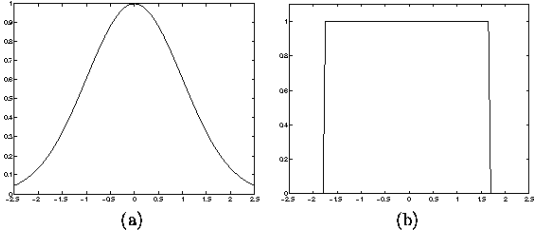
- The neighborhood function includes two parts.
  - A decreasing learning rate function $\alpha(t)$.
  - A function dictating the form of the neighborhood function, which also determines the rate of change around the winner unit m_c.
- The neighborhood function can be written as

$$h_{ci}(t) = \alpha(t) \cdot \exp(-\frac{||r_i - r_c||^2}{2\sigma(t)^2})$$

  in the case of the Gaussian neighborhood function around the winner neuron m_c. Here r_i is the position of the excited neuron i and r_c the position of the winning neuron. $\sigma(t)$ decreases over time.
- A variety of neighborhood functions can be used.

Neighborhood function

- A neighborhood function with a Gaussian kernel (a) around the winner neuron is computationally demanding as the exponential function has to be calculated, but can well be approximated by the ”bubble” neighborhood function.
- The bubble neighborhood function (b) is a constant function. Every neuron in the neighborhood is updated the same proportion of the difference between the neuron and the presented sample vector.
- The bubble neighborhood function is a good compromise between the computational cost and the approximation of the Gaussian.



BMU's neighborhood

- The neighborhood decreases over time.
- The figure is drawn assuming the neighborhood remains centered on the same node, in practice the BMU will move around according to the input vector being presented to the network.
- Over time the neighbourhood will shrink to the size of just one node, i.e., the BMU itself.

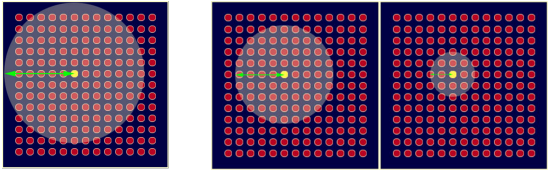


Figure: Initial radius, gradually shrinking radius

Learning rate

- Learning rate $\alpha(t)$ is a decreasing function of time t.
- Three learning-rate functions are common:
  - A linear function decreases to zero linearly during the learning from its initial value.

$$\alpha(t, T) = \alpha(0) \cdot (1 - t/T)$$

    where T is the total number of iterations (epochs), t a specific iteration.
  - An inverse function decreases rapidly from the initial value.

$$\alpha(t) = \alpha(0) \cdot 1/t$$

  - A power function

$$\alpha(t, T) = \alpha(0) \cdot e^{(-t/T)}$$

Learning rate

- The initial values for $\alpha(0)$ must be determined. Usually, when using a rapidly decreasing inverse α function, the initial values can be larger than in the linear case.
- The learning is usually performed in two phases. On the first round relatively large initial alpha values are used ($\alpha = 0.3, \cdots, 0.99$) whereas small initial α values ($0.1, \cdots, 0.01$) are used during the other round.
- This corresponds to first tuning the SOM approximately to the same space as the inputs and then fine-tuning the map. There are several rules-of-thumb by experiments for picking suitable values. See the monograph by Kohonen.

Learning rate examples

- (a) A linear function (b) An inverse function

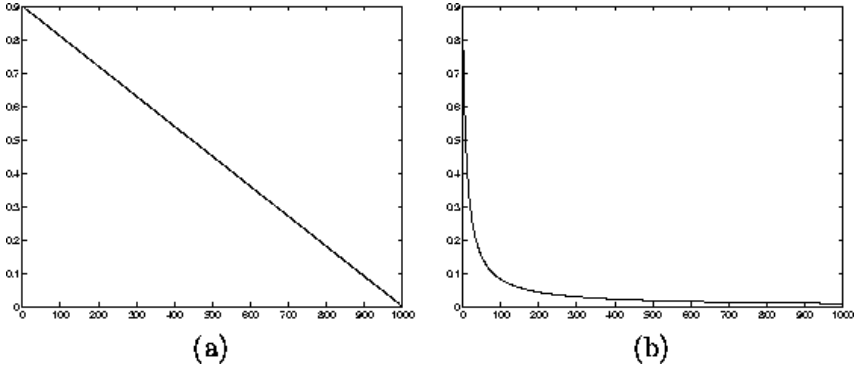
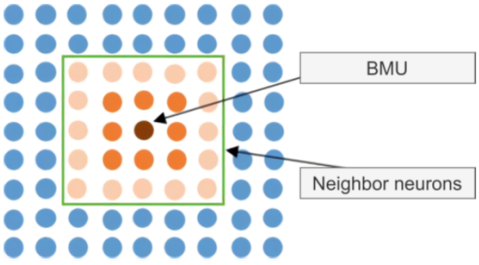
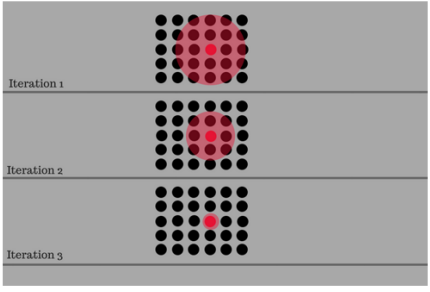


Figure: Linear rates as a function of time

Cooperation

- When the BMU is found, its own weights and its neighbor neurons' weights are updated together using the weight update formula, which returns updated weights for the next iteration.





Adaptation

- There are two hyper-parameters controlling the learning speed of the SOM network, including the learning rate and the neighbor size.
- The adaptation process aims to decrease these two hyper-parameters after each iteration to make the SOM learning converged.
- The adaptation of the parameters is done once for each iteration or epoch.

Comments

- There are a number of alternative equations used in the learning process of self-organizing maps.
- A lot of research tries to get the optimal values for the number of iterations, the learning rate, and the neighborhood radius.
- The inventor, Teuvo Kohonen, suggested that this learning process should be split into two phases.
  - Phase 1 Learning: The learning rate is reduced from 0.9 to 0.1 and the neighborhood radius from half the diameter of the lattice to the immediately surrounding nodes.
  - Phase 2 Fine-Tuning: The learning rate is further reduced from 0.1 to 0.0. However, there would be double or more iterations in Phase 2 and the neighborhood radius value should remain fixed at 1, meaning the BMU only.

Visualization

- U-matrix (unified distance matrix) representation of SOM visualizes the distances between the neurons, also called distance map.
- The distance between the adjacent neurons is calculated and presented with different colorings between the adjacent nodes.
- A dark coloring between the neurons corresponds to a large distance and thus a gap between the weight values in the input space.
- A light coloring between the neurons signifies that the weight vectors are close to each other in the input space.
- Light areas can be thought as clusters and dark areas as cluster separators. This can be a helpful presentation when one tries to find clusters in the input data without having any *a prior* information about the clusters.
- Teaching a SOM and representing it with the U-matrix offers a fast way to get insight of the data distribution.

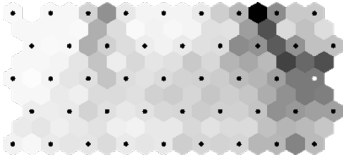


Figure: The U-matrix of a 9×5 hexagonal map

The neurons of the network are marked as black dots. The representation reveals that these are a separate cluster in the upper right corner of this representation. The clusters are separated by a dark gap.

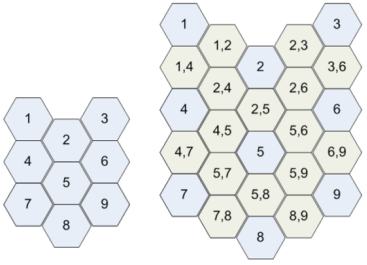
U-matrix: How to obtain?

A U-matrix is a visual representation of the distances between neurons in the input space.

- Step 1: calculate the distance between adjacent neurons, using their trained vector.
- Step 2: Assign a color to the distance value, e.g. a light gray to small value, and a dark gray to a large value, and the other values to corresponding shades of gray.
- Step 3. Use these colours to paint the cells of the U-matrix.

U-matrix example

- Assume a 3x3 hexagonal map.



A 3x3 hexagon SOM

Its U-matrix representation

- The $\{x, y\}$ elements are the distance between neuron x and y, and the values in x elements are the mean (or max or medium) of the surrounding values. For example, $\{4, 5\}$ = distance$(4, 5)$; $\{4\}$ = mean$(\{1, 4\}, \{2, 4\}, \{4, 5\}, \{4, 7\})$.
- To calculate the distance, use the trained n-dimensional vector of each neuron and the distance formula used for the training of the map (usually Euclidean distance).

U-matrix

- The U-matrix contains in each cell the euclidean distance (in the input space) between neighboring cells.
  - Small values mean that SOM nodes are close together in the input space, whereas larger values mean that SOM nodes are far apart, even if they are close in the output space.
  - Usually, those distance values are discretized, color-coded based on intensity and displayed as a kind of heatmap.
  - When such distances are depicted in a grayscale image, light colors depict closely spaced node weight vectors and darker colors indicate more widely separated node weight vectors. Thus, groups of light colors can be considered as clusters, and the dark parts as the boundaries between the clusters.
- For example, a 5×1 map: $m(1), m(2), m(3), m(4), m(5)$, where $m(i)$ denotes one map unit. The U-matrix is a 9×1 vector:
  $u(1), u(1,2), u(2), u(2,3), u(3), u(3,4), u(4), u(4,5), u(5)$ where $u(i,j)$ is the distance between map units $m(i)$ and $m(j)$, and $u(k)$ is the mean (or minimum, maximum or median) of the surrounding values, e.g. $u(3) = (u(2,3) + u(3,4))/2$.

U-matrix with variation

- Instead of hexagonal units, U-matrix may come in a 2D topology with rectangular units.
- The idea is the same, i.e., to calculate the distance and visualize a color map according to the distance values, but the exact distance calculation might be different from implementation to implementation.

U-matrix with variation

- The red box can be calculated by averaging the four neighbor units in yellow color.

1	2	3
4	5	6
7	8	9

Figure: A 3x3 map

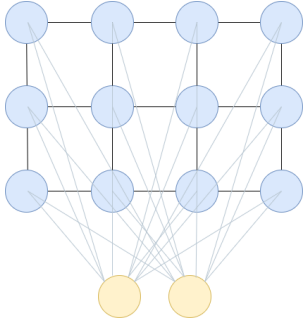
1	1,2	2	2,3	3
1,4	?	2,5	?	3,6
4	4,5	5	5,6	6
4,7	?	5,8	?	6,9
7	7,8	8	8,9	9

Figure: Its corresponding U-matrix

- Note: There are implementation libraries which present a U-matrix overlaid on its SOM without adding the units in yellow and red color.
- Attention: Need to check the implementation API to know exactly what the distance represents in a U-matrix.

Question: Why is SOM special?

- It is quite different from the mainstream so-called neural networks.
  - Output-layer neurons are different.
    - Are there any weighted sum calculations?
    - Is there any nonlinear transformation?
  - Each node (unit) in the output layer has a location with neigborhood.
  - Connections on the output layer are different.
  - More different? Is the learning process different, e.g. any error propagation, any gradient descent? Why?
  - Is the output layer different both in concept and dimensioning?
  - Does it make sense to add one or more hidden layers in SOM? Why?



Implementations of SOM

There are a few open SOM implementations in Python.

- sklearn-som `https://pypi.org/project/sklearn-som/`
- minisom `https://github.com/JustGlowing/minisom`
  Minisom playground: `https://share.streamlit.io/justglowing/minisom/dashboard/dashboard.py`
- SimpSOM
  GitHub: `https://github.com/fcomitani/SimpSOM`
  Sample: `https://www.kaggle.com/asparago/unsupervised-learning-with-som`
- somoclu
  Documentation:`https://peterwittek.github.io/somoclu/`
  Github: `https://somoclu.readthedocs.io/en/stable/`
  Example: `https://github.com/abhinavralhan/kohonen-maps/blob/master/somoclu-iris.ipynb`

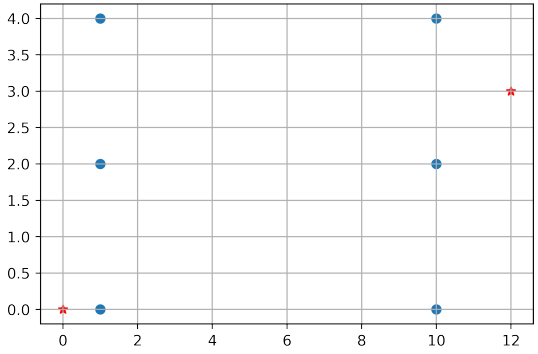
Application example

- Application domains
  - Clustering, classification
  - Visualization of high-dimensional data on 2D space: Reveal the clusters and topology of dataset on a 2D plane.
  - Vector quantization
  - ...
- Examples [a]
  - A simple clustering example
  - A slightly complicated example for clustering and visualization.

[a] Note: The code of the examples is only for teacher.

A simple clustering example

We use the same data set as that for K-means, we want to know: how and if SOM generates the same results?



http://localhost:8888/notebooks/IL2233VT22/Lec9_som/simple_som_example.ipynb

The Iris example again

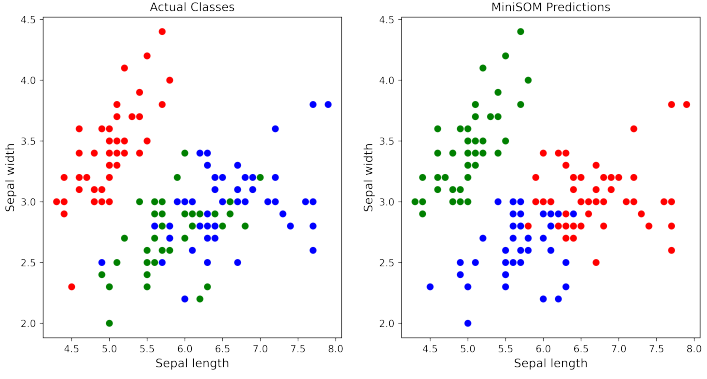
We use the **minisom** library to go through the example.

- Load the dataset from sklearn and visualize the data in 2 or 3 dimensions.
- Define a SOM (K-means) model with 3 clusters.
- Fit the 3-cluster model to the data set (you can try using 2 or 3 or all 4 features of the data set for clustering).
- Visualize the estimated cluster labeling and true cluster labeling
- Output cluster labels. Note the cluster labels likely need to be re-mapped.
- Generate a confusion matrix.
- Define a larger grid SOM, train the SOM to fit the data, and generate a grid-shape U-matrix (distance map).
- Define a hexagonal SOM, train, and generate a hexagonal U-matrix.

`http://localhost:8890/notebooks/IL2233VT22/Lec9_som/minisom-clustering-iris-umarix-square-polygon.ipynb`

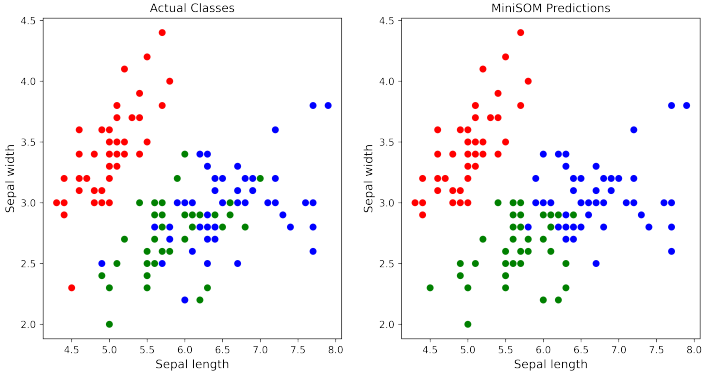
Visualization after clustering

We use the first two dimensions of data to visualize the data with 3 classes.



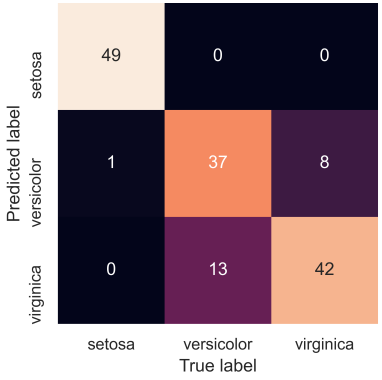
Visualization after clustering and label remapping

After label remapping, the 3 labels (0, 1, 2) /colors (r, g, b) match.



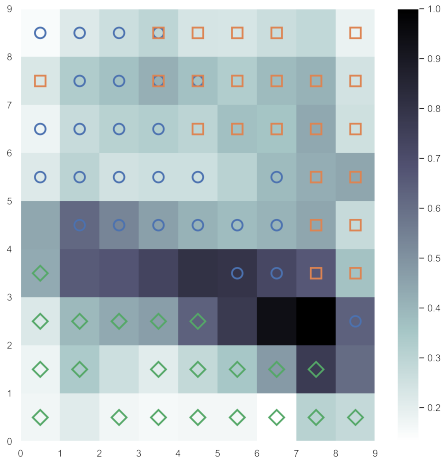
The confusion matrix

After label remapping, we can correctly compare the labels and calculate evaluation metrics such as accuracy, and show a confusion matrix.



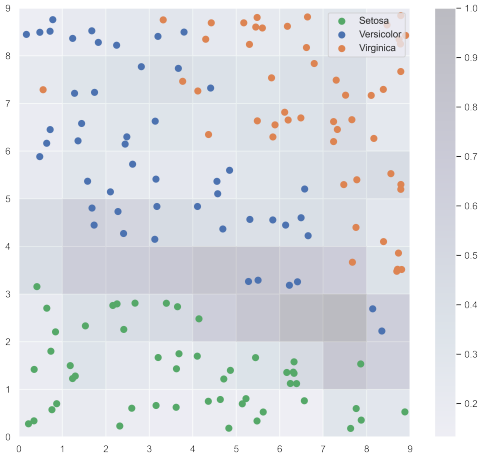
Mapping the data to a larger SOM

- A 9×9 SOM is trained and its U-matrix (distance map) is visualized using a heatmap.
- The neurons of the map are displayed as an array of cells and the color represents the (weights) distance from the neighbour neurons.
- On top of the color map, the markers represent the samples mapped in the specific cells.



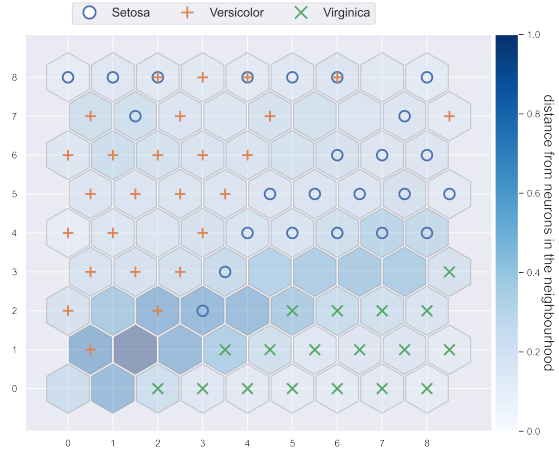
U-matrix with mapped data points

- To see how the samples are distributed across the map, we can draw a scatter graph where each dot represents the coordinates of the winning neuron.
- A random offset is added to avoid overlaps between points within the same cell.



U-matrix in hexagonal cells

We first train a hexagonal SOM and draw its u-matrix.



Summary

- An SOM is made up of a grid (commonly rectangular or hexagonal) of nodes, where each node contains a weight vector that is of the same dimensionality as the input vector.
- The nodes may be initialized randomly, but an initialization that roughly approximates the distribution of the dataset will tend to train faster.
- When applied at a smaller scale (smaller grid), the SOM algorithm behaves similarly to k-means clustering. At a larger scale, SOMs reveal the topology of complex datasets in a powerful way.

Summary

- The training algorithm iterates as observations are presented as input.
  - Identifying the winning node in the current configuration—the Best Matching Unit (BMU). The BMU is identified by measuring the Euclidean distance in the data space of all the weight vectors.
  - The BMU is adjusted (moved) towards the input vector.
  - Neighboring nodes are also adjusted, usually by lesser amounts, with the magnitude of neighboring movement being dictated by a neighborhood function e.g. a Gaussian neighborhood function.
- This process repeats over potentially many iterations with decreasing learning rate and neighborhood size, using sampling if appropriate, until the network converges (reaching a position where presenting a new input does not provide an opportunity to minimize loss.)
- The inference is simple: Given a new input vector, find its BMU in the output layer.
- The U-matrix is a powerful tool to visualize its topology-preserving property mapping possibly high-dimension inputs to units on a 2D space.

References

- Teuvo Kohonen. Self-Organizing Maps. Springer, Berlin, Heidelberg, 1995. 2001 (third edition)
- Teuvo Kohonen, Erkki Oja, Olli Simula, Ari Visa, and Jari Kangas. Engineering applications of the self-organizing map. Pro. Of IEEE
- Jaakko Hollmén Process Modeling Using the Self-Organizing Map, 1996. https://users.ics.aalto.fi/jhollmen/dippa/node9.html
- A. Ultsch and H.P. Siemon. Kohonen's self organizing feature maps for exploratory data analysis. In Proc. INNC'90, Int. Neural Network Conf., pages 305-308, Dordrecht, Netherlands, 1990. Kluwer.
- N. Q. Hoan, "Improving feature map quality of SOM based on adjusting the neighborhood function", International Journal of Computer Science and Information Security (IJCSIS), vol. 14, no. 9, 2016.
- Rubik. Introduction to Self-Organizing Maps. August 2018. https://rubikscode.net/2018/08/20/introduction-to-self-organizing-maps/
- A website with SOM implementation code: http://www.ai-junkie.com/ann/som/som1.html