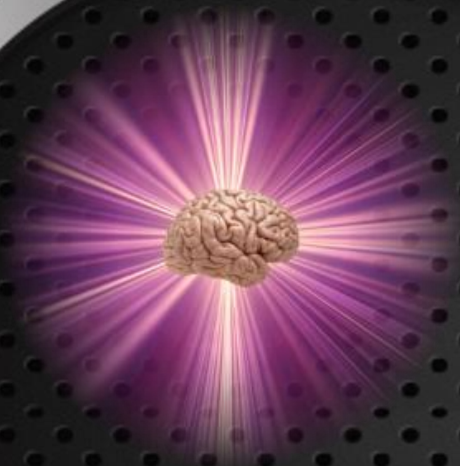
AGH University of Science and Technology
Faculty of Electrical Engineering, Automatics, Computer Science and Biomedical Engineering
Department of Biocybernetics and Biomedical Engineering



Computational Intelligence

Unsupervised Learning, Autoencoders, and Clustering Algorithms

Adrian Horzyk
horzyk@agh.edu.pl

Google: Adrian Horzyk

Machine Learning Approaches

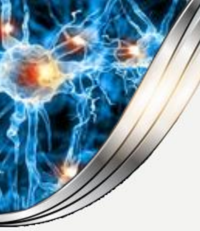
How to train networks on different data

and what they can be trained?

Machine Learning

Machine learning:

- is the science and art of programming computers (part of artificial intelligence) so they can learn from data (examples) called the training set;
- "the field of study that gives computers the ability to learn without being explicitly programmed" (Arthur Samuel, 1959);
- is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead;
- "A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E." (Tom Mitchell, 1997)
- Example of a "spam and ham" task: Your spam filter is a machine learning program that can learn to flag spam if we give it examples of spam emails (e.g., flagged by users) and examples of regular (no-spam, also called "ham") emails. Each training example is called a training instance or sample.
  The task T is to flag spam for new emails, the experience E is the training data, and the performance measure P needs to be defined, e.g., by the ratio of correctly classified emails. This particular performance measure is called accuracy, and it is often used in classification tasks.



Machine Learning Strategies

Main groups of machine learning strategies are:

- **Unsupervised learning** – groups data according to their similarity.
- **Supervised learning** – makes predictions about data or classify data.
- **Semisupervised (weakly-supervised) learning** – uses partially labelled data for supervised learning based on unsupervised components like Restricted Boltzmann Machines (RBM).
- **Reinforcement learning** – interacts with the environment and maximizes a cumulative reward that controls a training process where data are sequential in time. It is an area of machine learning concerned with how agents ought to take actions in the environment so as to maximize a cumulative reward.
- **Motivated learning** – defines fundamental needs and automatically develops secondary needs which affect the fundamental ones and control the interactions with the environment. During the learning process, fundamental needs should be satisfied what minimize pain (a penalty) and maximize pleasure (a reward) – they work as motivating factors.
- **Associative learning (cognitive learning)** – aggregates the representation of similar features and objects, links them due to their real relations and actions of various kinds, connects them with different strengths and allows to trigger created associations recalling back related objects for a given context in time.

Unsupervised and Competitive Learning

How can we train a model without labels or predefined desired output values?

Unsupervised Learning

Unsupervised learning is a kind of machine learning that uses input data (vectors or matrices) to self-organize, group, or cluster them.

During this learning process, we try to use similarities between input objects (points in a hyperspace of input data), which allow achieving this goal.

Training data $\mathbb{S} = \{X^1, \dots, X^N\}$ are here not labelled, classified, or categorized where $X^n = [x_1^n, \dots, x_K^n] \in X$, and we have no strictly defined goal of the learning process in terms of results that should be achieved.

It is also associated with hierarchical clustering, k-means, self-organizing maps, neural gas, Hebbian learning, autoencoders, deep belief nets, generative adversarial networks (GAN).

The goal of unsupervised learning is to better understand data, their groups, local densities, find clusters and relations between the data which differentiate them.

Competitive Learning

Competitive learning is a kind of unsupervised learning in which nodes (or neurons) compete for the right to respond to a subset of the input data and adapt parameters (e.g. weights) to support such responses in the future.

Two major competitive strategies are:

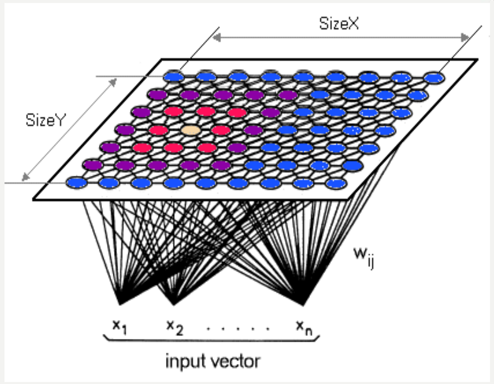
- winner-takes-all – where only a single node wins and adapts itself (weights), and
- winner-takes-most - where the winning node adapts the most, but the other nodes representing similar patterns are adapted as well with the diminishing gradients.

The algorithms like **vector quantization** and **Kohonen's self-organizing maps (SOM)** apply the competitive unsupervised principle.

Self-Organizing Maps (SOM) and Unsupervised Competitive Learning

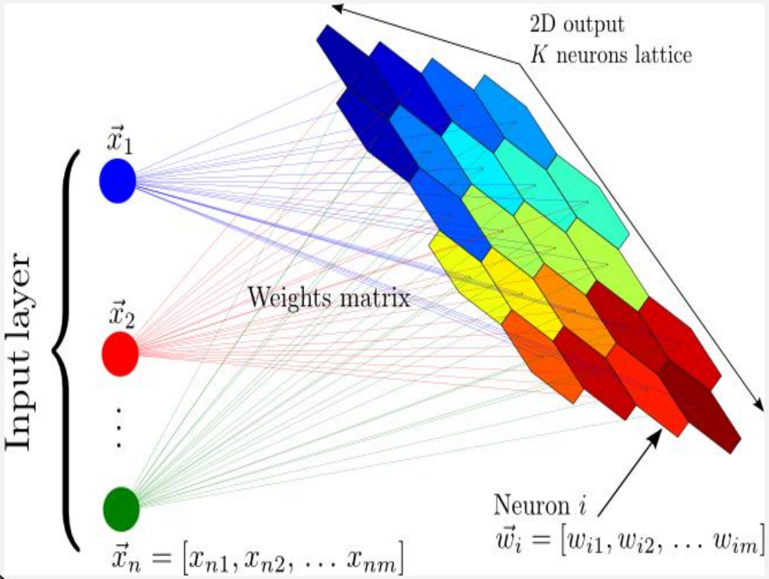
How to self-organize data using competition?

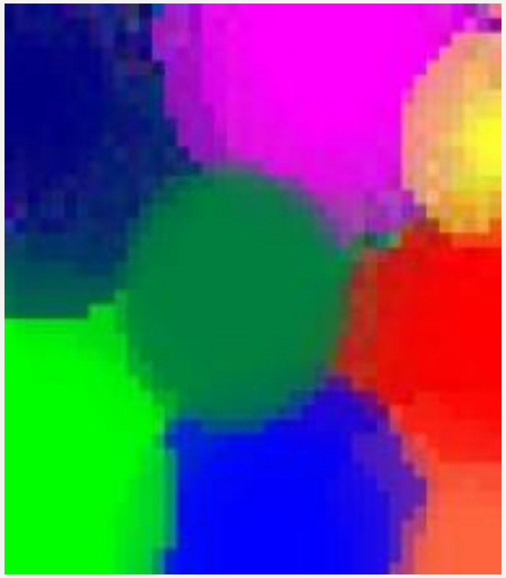
Self-Organizing Maps (SOM)

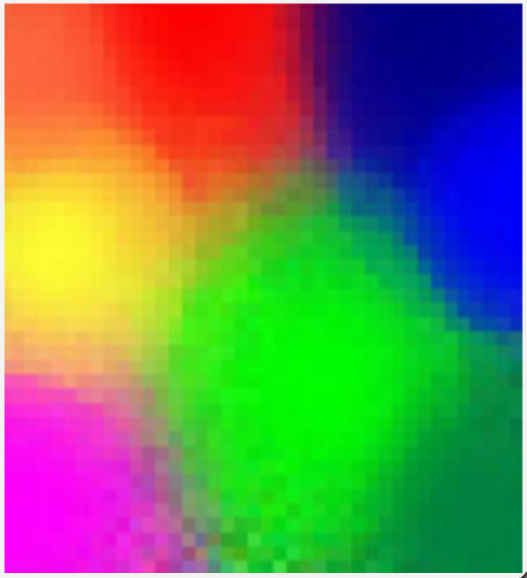


Kohonen's SOM (invented in 1982 by Tuevo Kohonen):

- is a network with a regular structure consisting of nodes that represent similar objects from the input data space;
- represents multidimensional data in a smaller space (usually 2D or 3D), which does not always allow for the representation of groups of similar objects by neighboring nodes;
- the nodes in close proximity should represent similar groups (clusters) of objects:


2D output
K neurons lattice
Input layer
$\vec{x}_1$
$\vec{x}_2$
Weights matrix
$\vec{x}_n = [x_{n1}, x_{n2}, \dots x_{nm}]$
Neuron i
$\vec{w}_i = [w_{i1}, w_{i2}, \dots w_{im}]$





It is impossible to map the 3D space of RGB colors in 2D space, representing all similar colors side by side: The yellow should be between red and green. The pink should be between red and blue. The blue should be between green and blue.

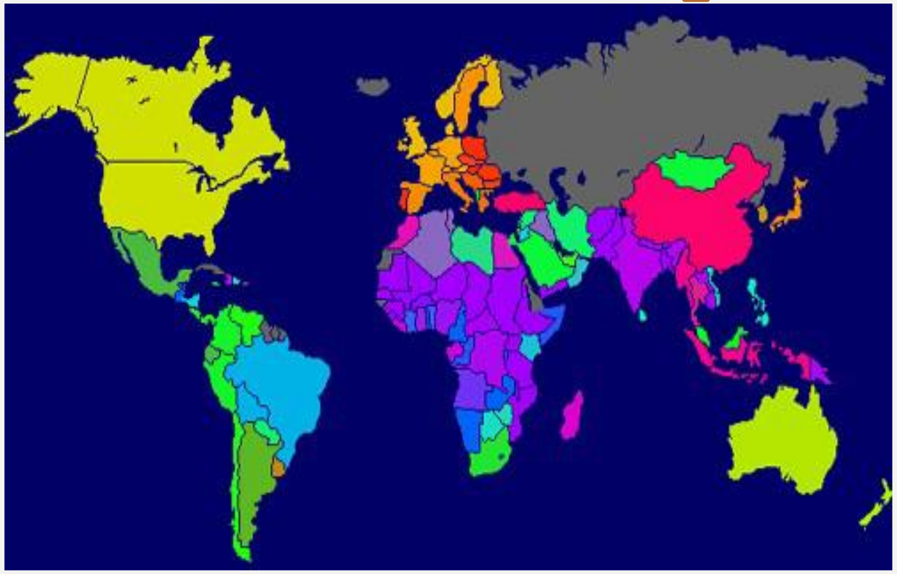
Self-Organizing Maps (SOM)

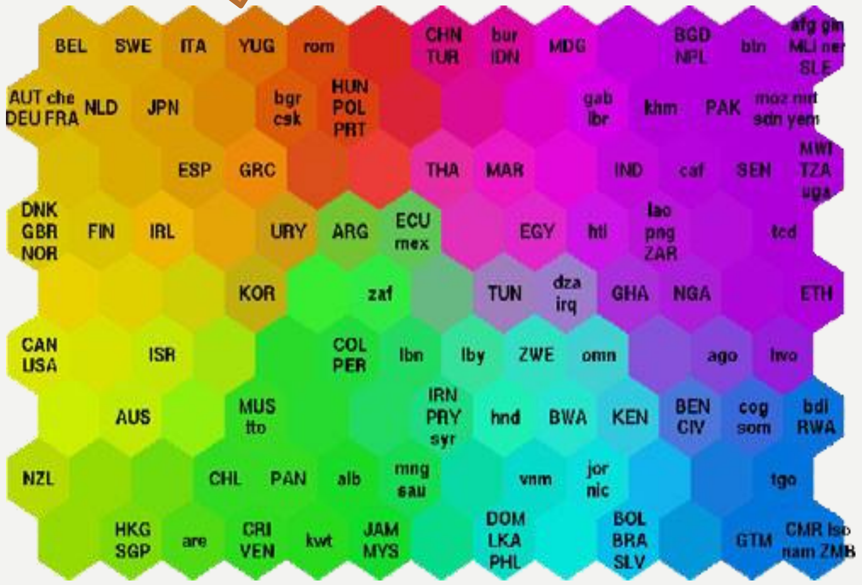
Kohonen's SOM (invented in 1982 by Tuevo Kohonen):

- Is adapted with the use of **unsupervised competitive learning**, i.e. trained without a teacher/supervisor (indication of the purpose of learning, i.e. class, predicted values) in such a way that **the nodes compete** with each other for the representation of input data;
- **Neighborhood of nodes** plays a significant role during the adaptation of the network, as it also allows for the adaptation of nodes close to the winner leading to the automatic reorganization of representation of objects in this network during the training process;
- They resemble neural networks to a small extent, as inputs to nodes of this network **are not weighed**, but compared to weight vectors based on an distance (e.g. Euclidean);
- They determine the average of the represented patterns at each node of the network stored in the weight vector, where the patterns are represented in the node, which cause that the given node becomes the winner (it is the closest to the presented input).

Sample SOM Network

The map showing the quality of life in different countries of the world used for the adaptation of the SOM network gave the following result aggregating the representation of different countries and placing other countries with a similar quality of life in their neighborhood, e.g.:

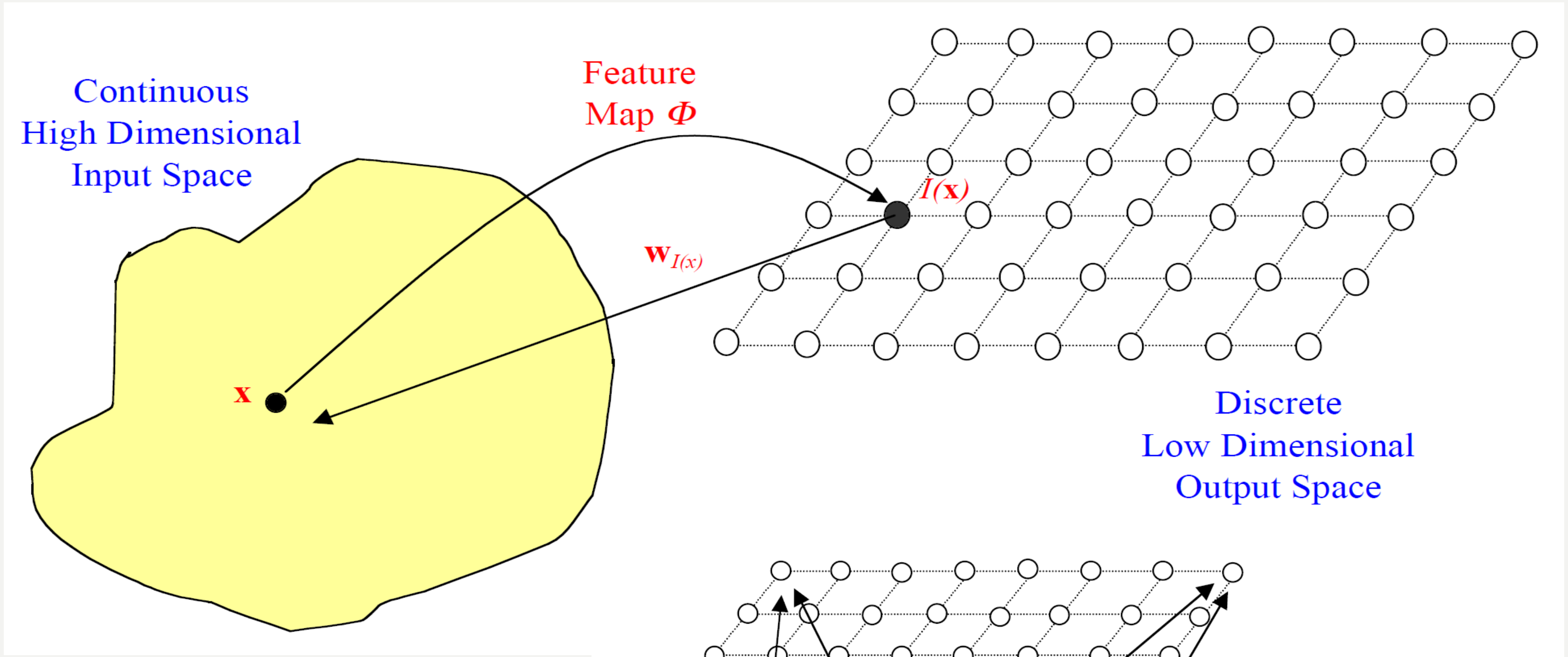




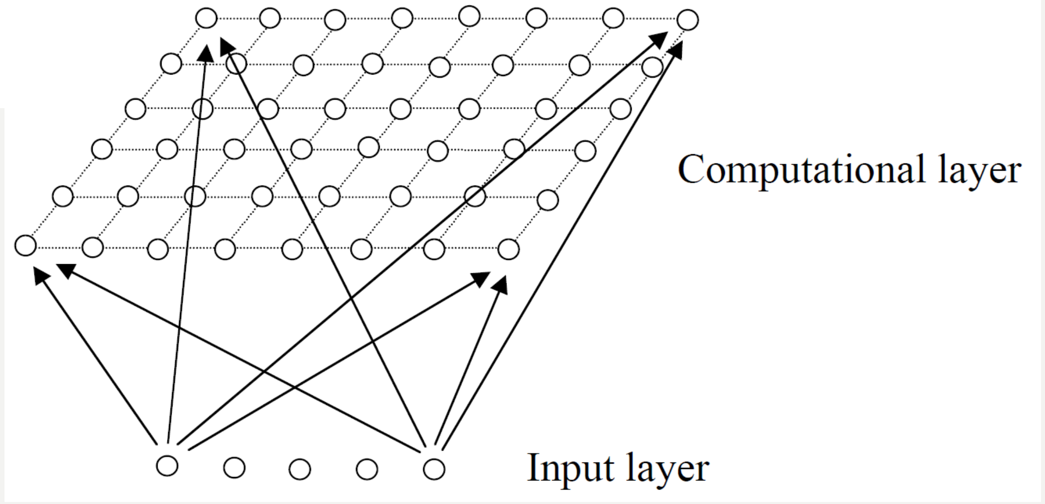
- Poland (POL) is represented by the same node as Hungary (HUN) and Portugal (PRT), on the left we see a close node representing the Czech Republic (CSK).
- Developed countries with a higher life quality, i.e. USA and Canada, are located in yellow nodes, while western European countries, i.e. Germany (DEU) and France (FRA) are represented by one yellow-orange node.

Creating a Map for Training Data

The x points from the input space are mapped to points I(x) in the output space.



Next, all inputs are connected to all points (SOM nodes):

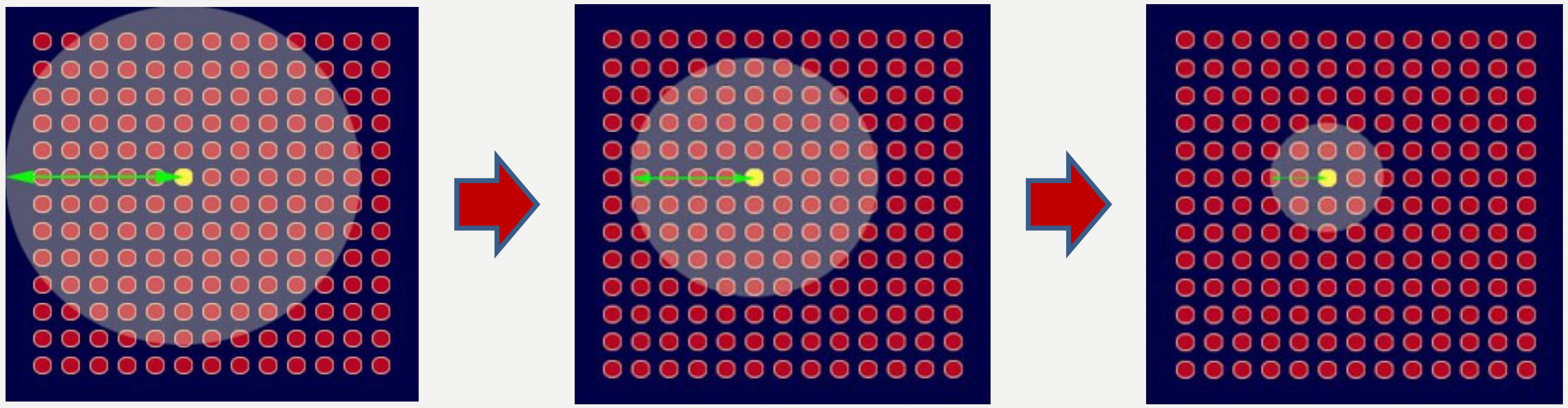


Competitive Training of SOM

The SOM network, based on the similarity of input patterns, updates the weights of the network nodes on **the competitive learning basis** in such a way that the node representing the value closest to the input data (the most similar) in the sense of the adopted metric (e.g. Euclidean metric) becomes the winner.

The winning node weights are most strongly updated towards the input data (i.e. reducing the distance) compared to neighboring patterns, which weights are updated weaker, the less the more away from the winner are.

By modifying the weights of neighboring nodes, we obtain the possibility of **representing similar sample groups by neighboring SOM nodes**:



Competition between SOM Nodes

Let us consider learning patterns in the form of vectors $X_k = \{x_1, x_2, \ldots, x_n\}$, which elements are represented by n input nodes of the SOM network.

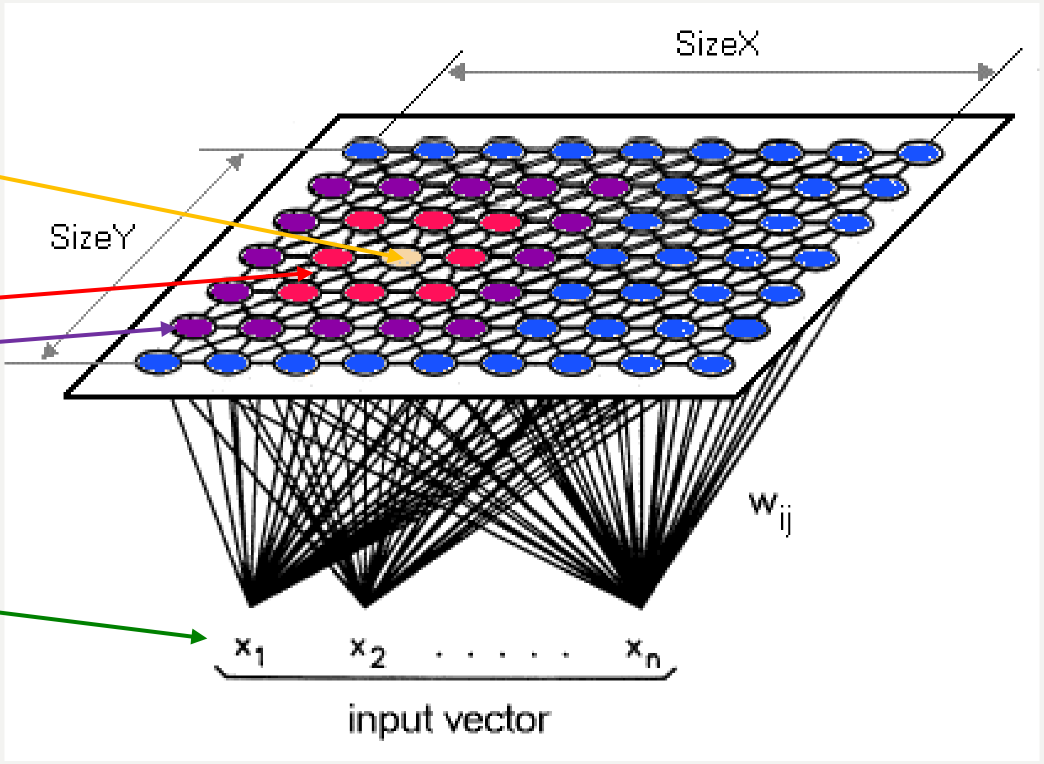
Each input node is associated with each node on the output map.

We initiate the weights with random values, so that the nodes react the strongest to the different combinations of input data $X_k = \{x_1, x_2, \ldots, x_n\}$.

Winner

Neighboring nodes

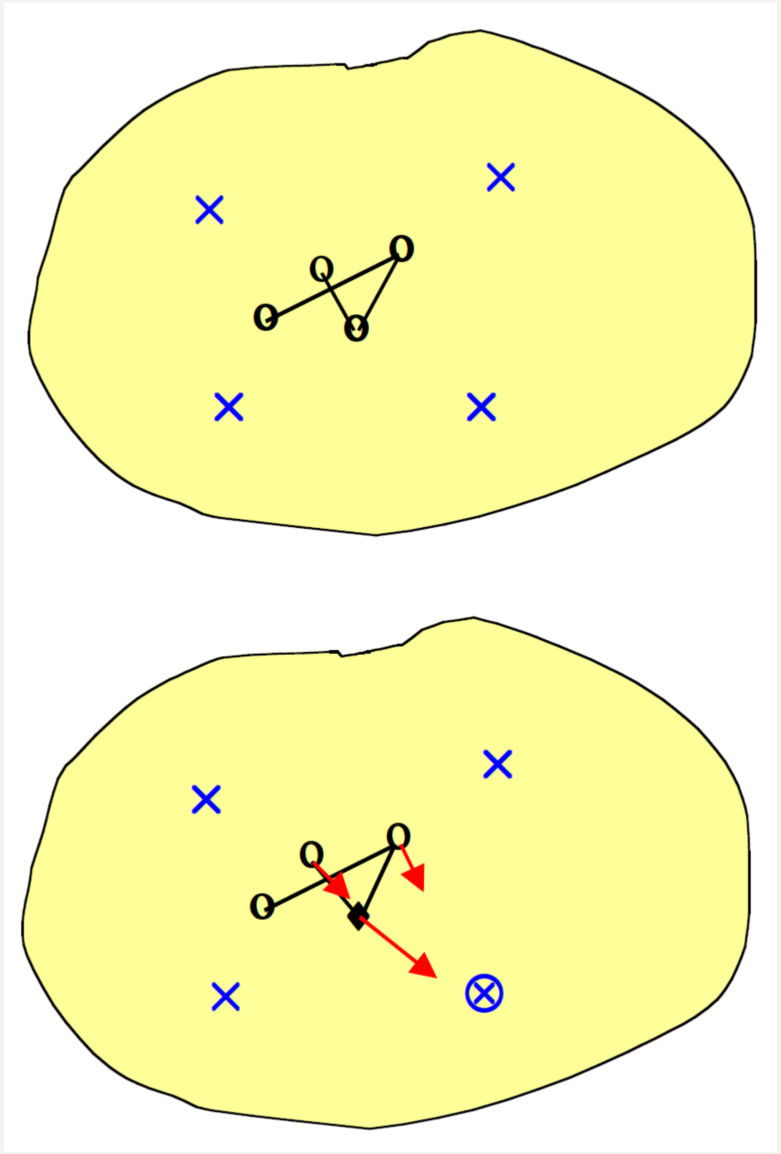
Input nodes



Competitive Training of SOM

Suppose we have four data points (crosses) in our continuous 2D input space, and want to map them onto four points in a discrete 1D output space. The output nodes map to points in the input space (circles). Random initial weights start the circles at random positions in the center of the input space.

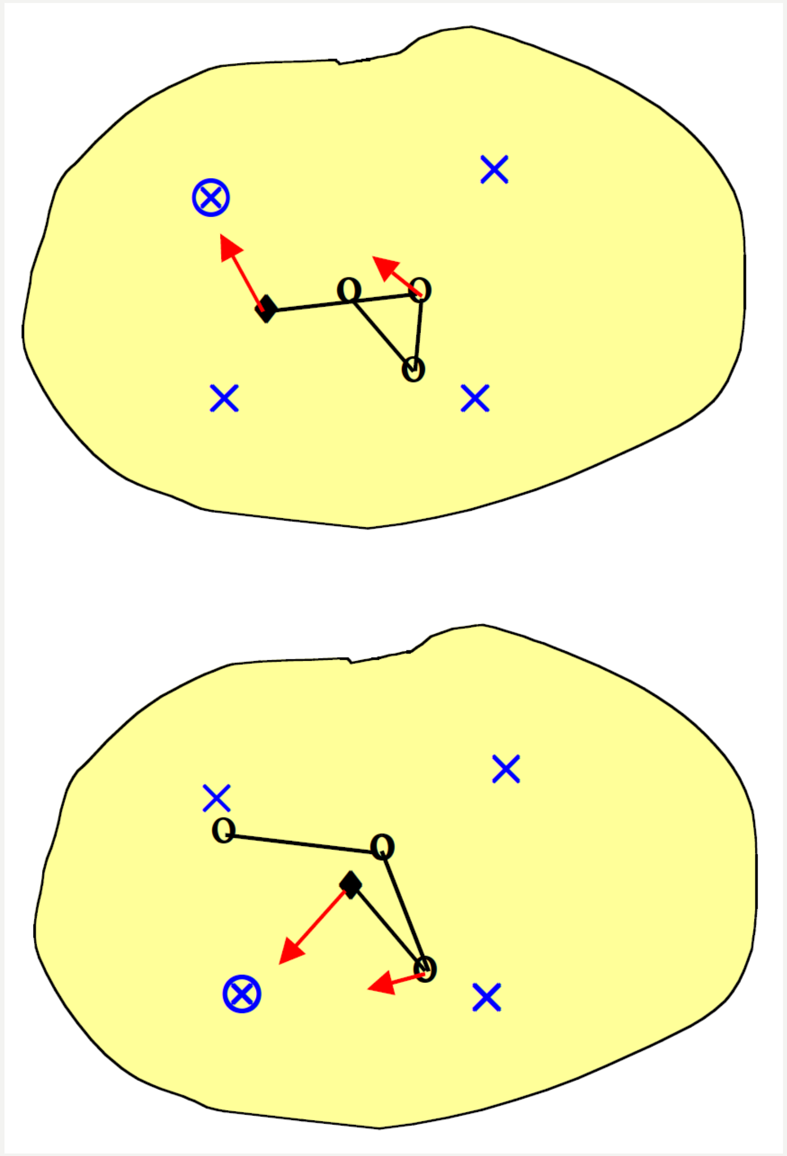
We randomly pick one of the data points for training (cross in circle). The closest output point represents the winning node (solid diamond). That winning node is moved towards the data point by a certain amount, and the two neighboring nodes move by smaller amounts (small arrows).



Competitive Training of SOM

Next, we randomly pick another data point for training (cross in a circle). The closest output point gives the new winning node (solid diamond). The winning node moves towards the data point by a certain amount, and the one neighboring node moves by a smaller amount (arrows).

We carry on randomly picking data points for training (cross in a circle). Each winning node moves towards the data point by a certain amount, and its neighboring node(s) move by smaller amounts (small arrows). Eventually, the whole output grid unravels itself to represent the input space.



Autoencoders

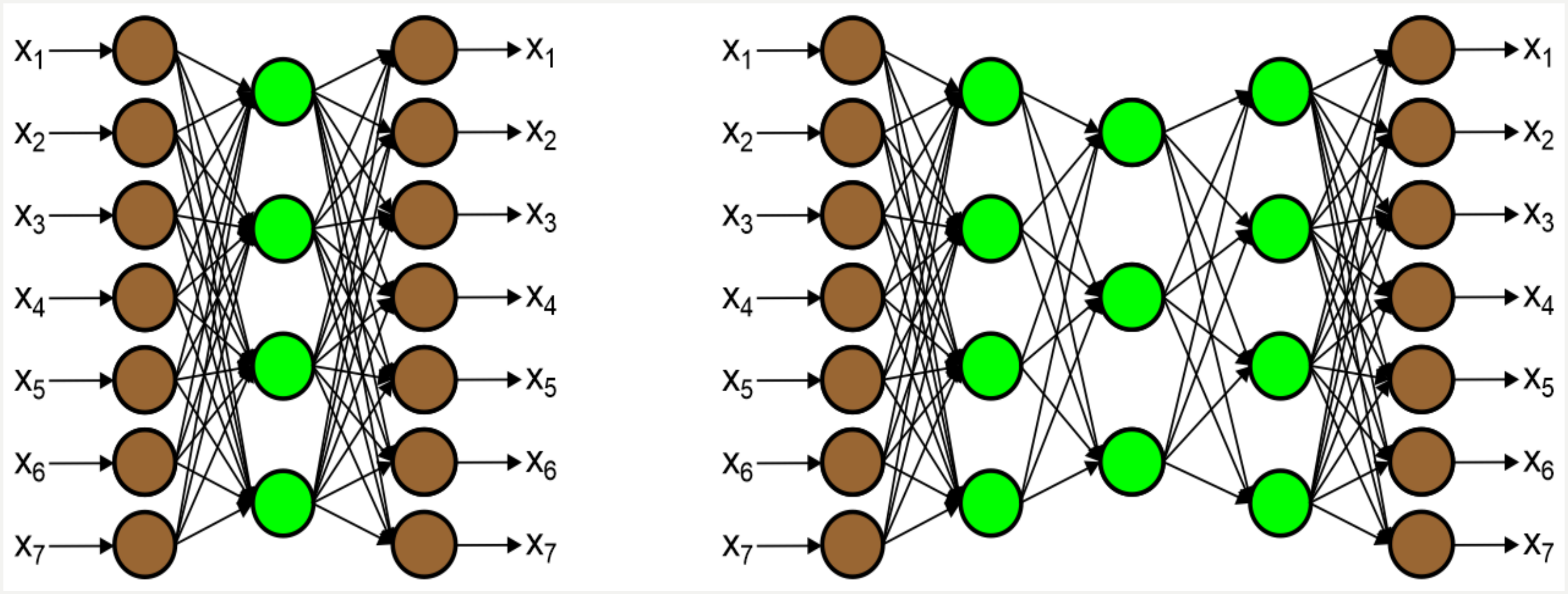
Why we use autoencoders?

Autoencoders

Autoencoders are a type of artificial neural networks used for unsupervised learning, self-association, anomaly detection, frequent feature representation, and efficient coding. They use a set of input patterns to teach identical outputs.

The primary goal of such training is to find a reduced number of neurons in the hidden layer(s) in relation to the input data dimension, which allows the input pattern to be reproduced without meaningful distortions.

This allows for the dimension reduction and such a representation of the set of input patterns that forces the creation of a representation of the features:

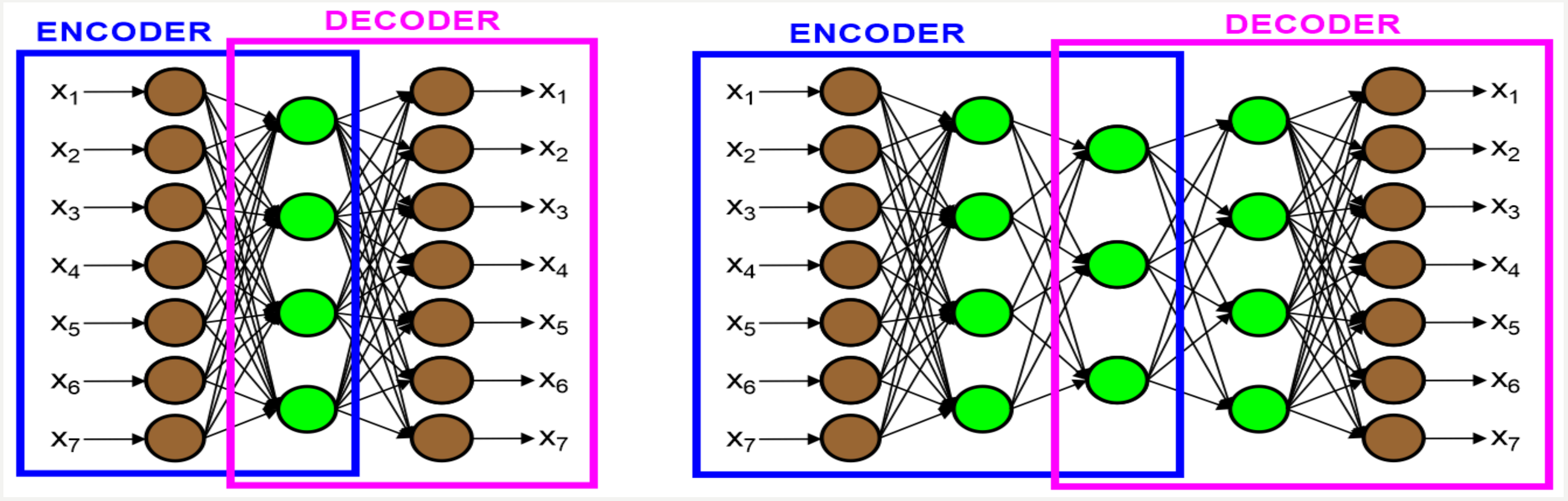


Autoencoders

Autoencoders can be trained with gradient methods used in supervised learning because the outputs are identical to the network inputs, so we can easily calculate the output error and then use, e.g., the error backpropagation method to adapt the weights of the autoencoder network. The essential question is why teach the network of self-association - that is, projection of identity?

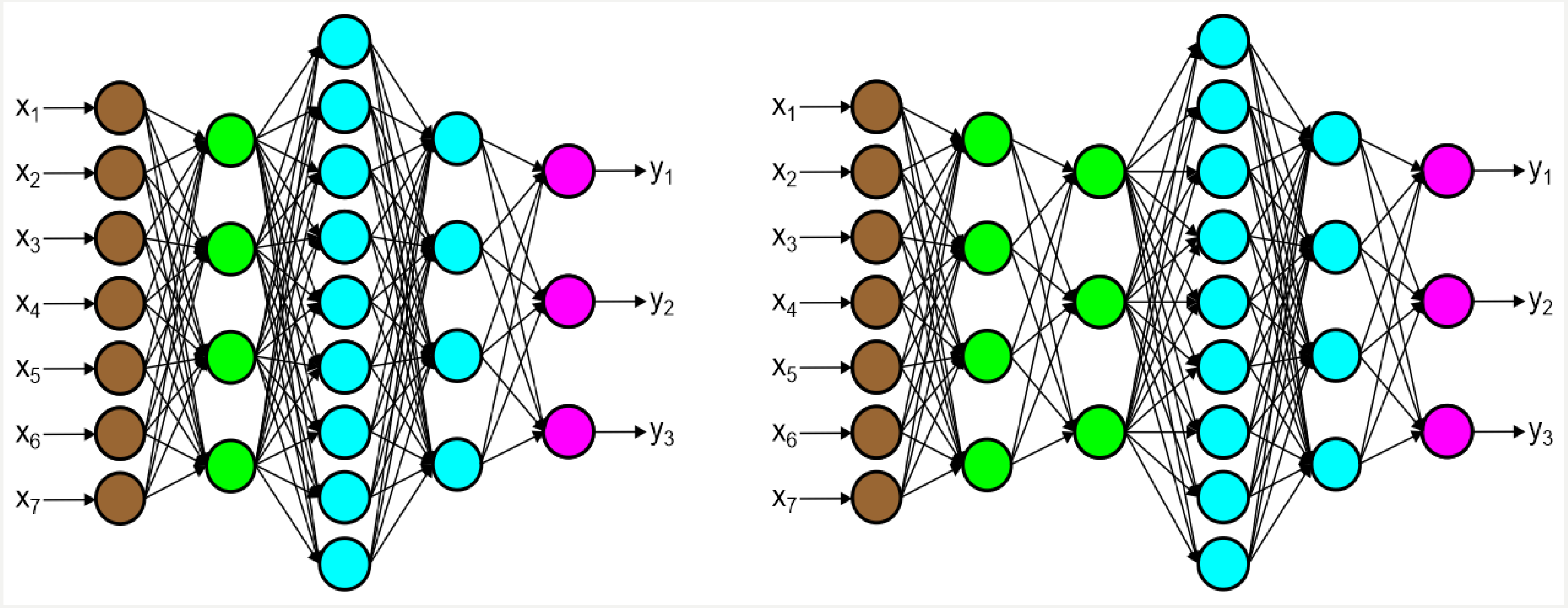
The main reason is that neurons in the hidden layer learn to sparingly represent the inputs, so that they can recreate it again. For this to be possible, they must represent certain common and similar features of the learned patterns.

We force this network to create a representation of the features of the learning examples. So there is a neural compression of the pattern representation.



Autoencoders

Autoencoders are often used in deep networks as non-supervised learning layers to determine the essential characteristics of the training patterns that can be further used for classification by cutting off the last layer(s) of such autoencoders:

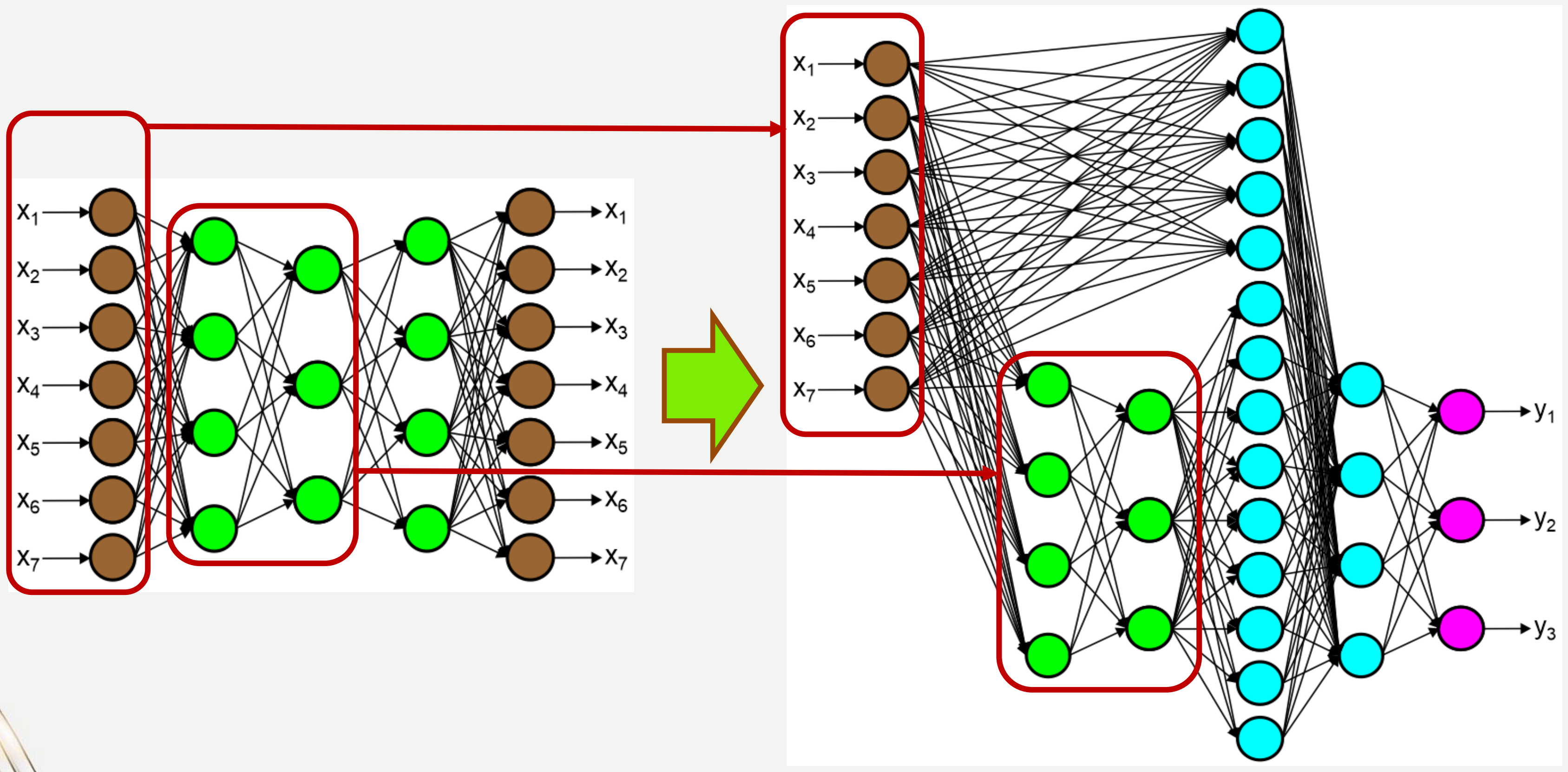


Autoencoders are first learned, then we cut off the last layer(s) and use the rest of the network (input layer and hidden layers) to further build a deep network, which is supervised learning in the final layers and can also fine-tune the encoder layers:

Hybrid Autoencoder Architectures

Autoencoders can be used in various hybrid neural network architectures for feature extraction, as can the first layers of CNN convolutional networks.

Various classification and recognition tasks require qualitative extraction of features (strong, invariant and well-discriminatory), as further results of classification or recognition depend on the quality of these features!



Clustering and Grouping

How can we use similarity to group objects?

Clustering and Grouping

Clustering

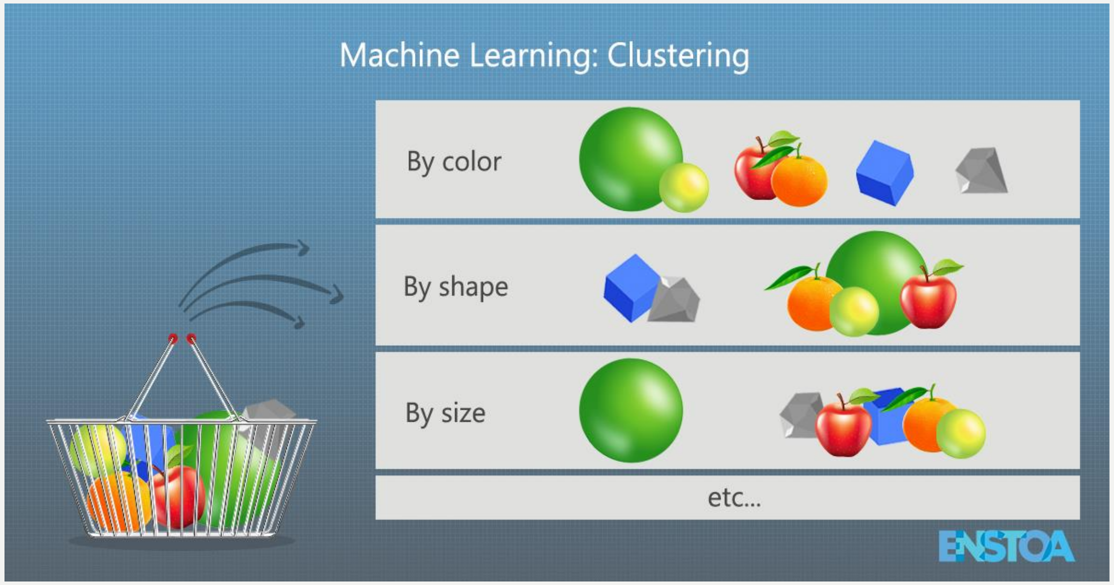
- is a data mining technique of a group of machine learning methods;
- involves the grouping of data points (objects) according to their similarity in an input data hyperspace;
- use unsupervised learning, so it does not require input data to be labeled as in the classification tasks;
- can group data into various groups, and next, these groups can be labeled;

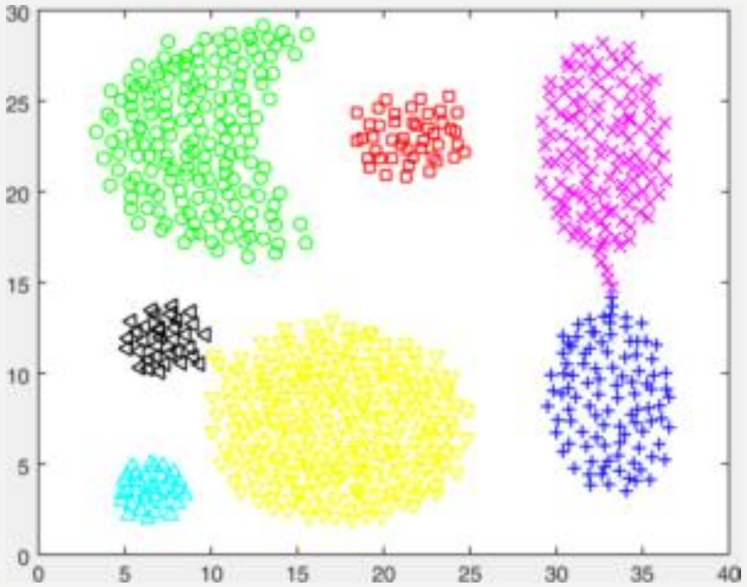
Main rule of clustering: Data points of the same cluster (group) should be similar, i.e. have similar features, while data points in different clusters (groups) should be different, i.e. their features should differ as much as possible.

Clustering methods can be:

- Strong (exclusive grouping) – data points from different clusters must be separable, i.e. each belongs to only a single cluster (clusters cannot overlap);
- Weak (non-exclusive grouping) – clusters can share some data points (clusters can overlap).

Clustering and Grouping





During clustering, selected features can be taken with different priorities.

These selected features are usually numerical in nature, so they are comparable and may indicate similarity between objects, or lack thereof.

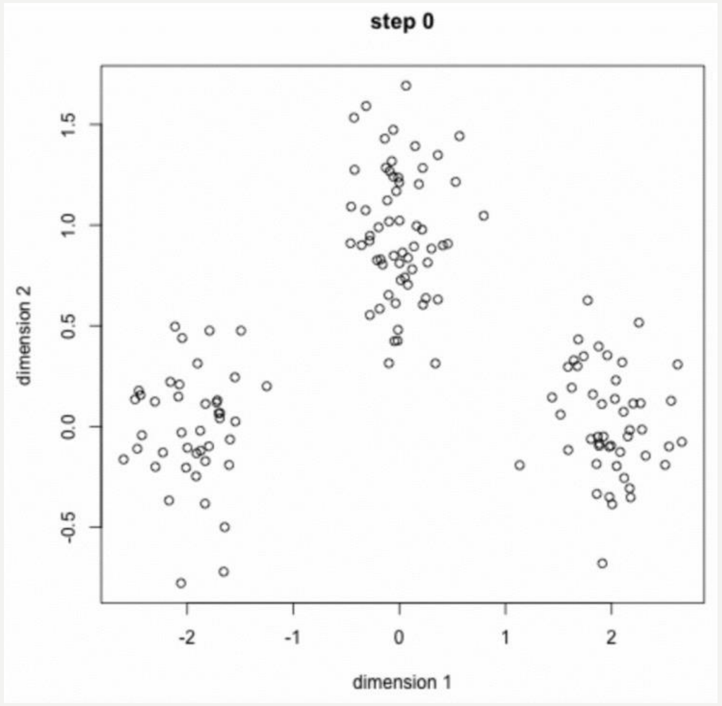
Clustering is the basic process responsible for the formation of knowledge in our mind as well as in computational and artificial intelligent systems.

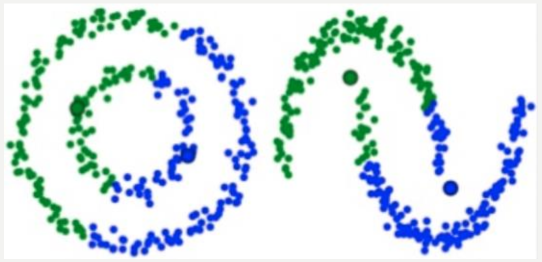
It can be used as a first step in the process of searching for frequent similar patterns that might be used for classification, regression or the other tasks.

K-Means Clustering

K-Means Clustering

- is probably the most well-known clustering algorithm;
- is easy to understand and implement;
- requires to set up the number of clusters first;
- produces different results according to the starting cluster centers;
- fails in cases where clusters are not circular.





K-Means Algorithm

1. Select a number of clusters (k) and randomly initialize their respective center points that are vectors of the same length as data point vectors.
2. Each data point is encompassed to the cluster point (group center) that distance to this data point is the smallest, i.e. each data point is assigned to the closest cluster (its center).
3. Based on these assigned points, we compute the group center by taking the mean of all the points in the group.
4. Repeat these steps until the group centers do not change much between subsequent iterations.

You can also randomly initialize the group centers a few times, and then rerun the algorithm to find the best clusters.

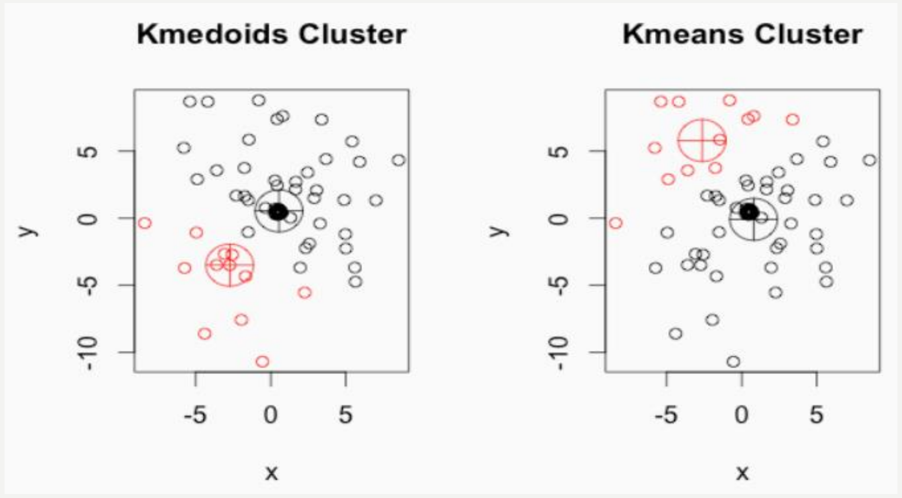
K-Median and K-Medoid Clustering

K-Medians Clustering

- is very similar to K-Means Clustering;
- computes the cluster centers using the medians instead of the means;
- is much less sensitive to outliers (if not too many of them);
- is much slower because it requires sorting of cluster points to compute medians.

K-Medoids Clustering

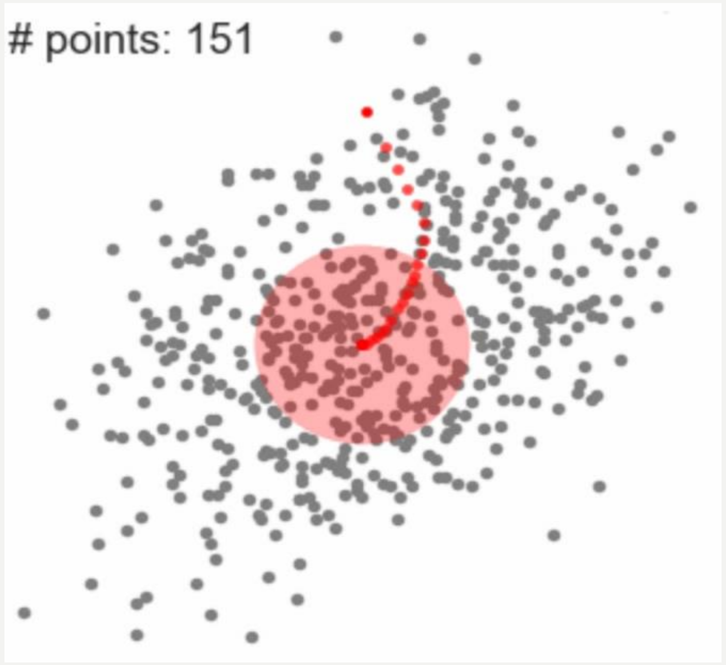
- is based on medoids (which are points that belong to the dataset) calculated by minimizing the absolute distance between the points and the selected centroid, rather than minimizing the square distance.
- is more robust to noise and outliers than k-means.

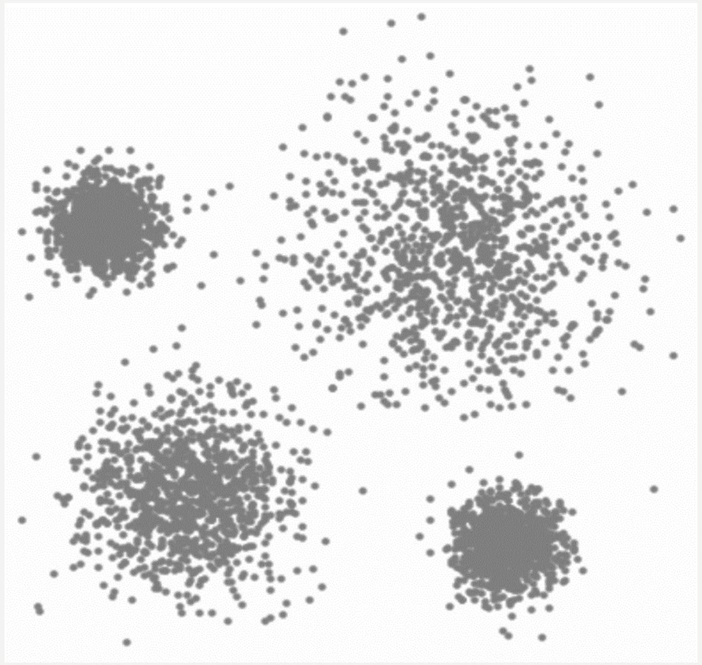


Mean-Shift Clustering

Mean-Shift Clustering

- Is a hill-climbing and sliding-window-based algorithm that attempts to find dense areas of data points in each step until convergence.
- Is trying to locate the center points of clusters by updating candidates for center points to be the mean of the points within the sliding window. Means automatically attract the sliding windows.
- These candidates are then filtered to eliminate near duplicates, forming the final set of center points (red dots) and their corresponding clusters. Each black dot in the animation below represents the centroid of a sliding window and each gray dot is a data point.
- The selection of the window radius of clusters is a non-trivial task.

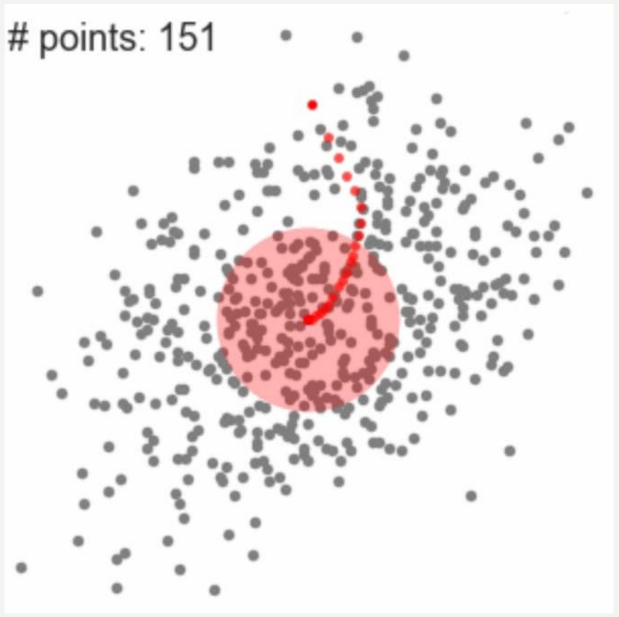


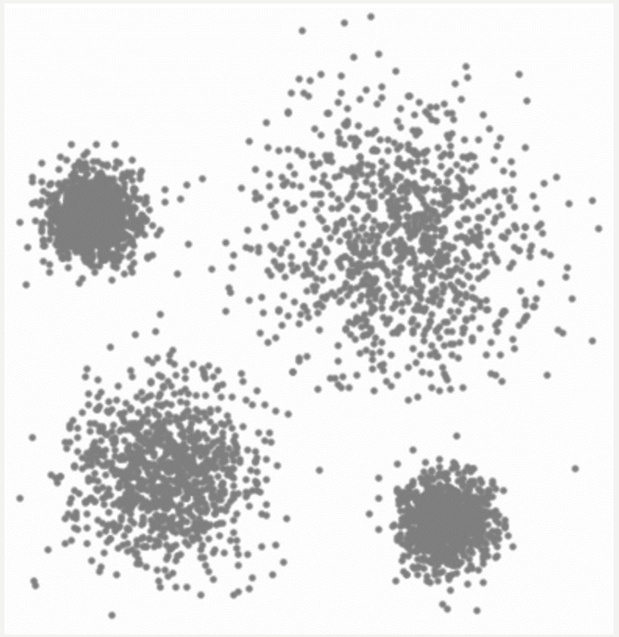


Mean-Shift Clustering

Mean-Shift Algorithm

1. Begin with a circular sliding window centered at a randomly selected point C having a radius r as the kernel.
2. In every iteration, the sliding windows are shifted towards regions of higher density by shifting the center point to the mean of the points within the window. The density within the sliding window is proportional to the number of points inside it.
3. Continue shifting the sliding of the windows according to the means (red points) until there are no directions at which they can be shifted and accommodate more points inside the kernels. In the figure, keep moving the circle until we can no longer increase the density, i.e. the number of points in this window.
4. Repeat this process until all points lie within windows. When multiple sliding windows overlap, the window containing the most points is preserved. The data points are then clustered according to the sliding window in which they reside.



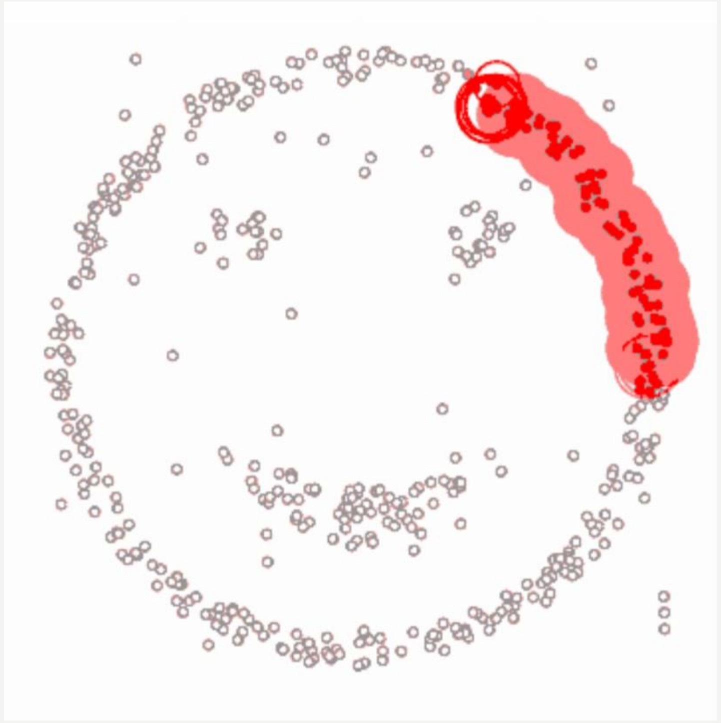


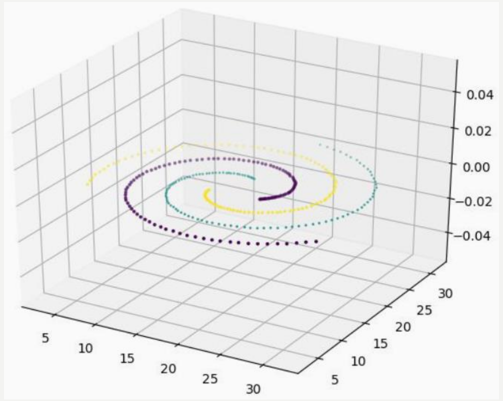
Density-Based Spatial Clustering

DBSCAN poses many great advantages over other clustering algorithms:

- It does not require a preset number of clusters.
- It identifies outliers as noise.
- It finds arbitrarily sized and arbitrarily shaped clusters quite well.

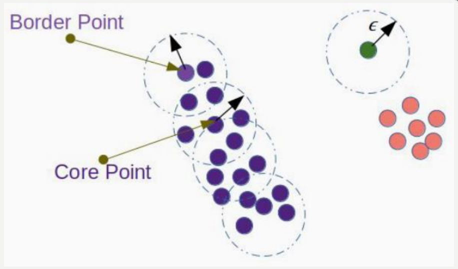
The main drawback of DBSCAN is that it does not perform as well as other algorithms when the clusters are of varying density.
This is because the setting of the distance threshold ε and minPointsNumber for identifying the neighborhood points will vary from cluster to cluster when the density varies.
This drawback occurs with high-dimensional data since again the distance threshold ε becomes challenging to estimate.



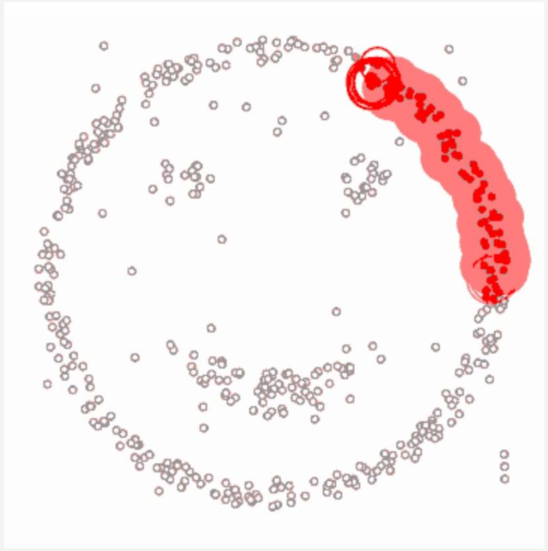


Density-Based Spatial Clustering

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a density-based clustering algorithm:



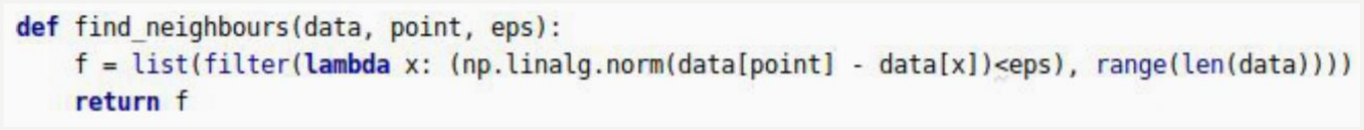
1. Begin with an arbitrary starting data point that has not yet been visited. Consider a neighborhood of this point defined by the circle with radius ε:
2. If there are a sufficient number of points (more than minPointsNumber) within this neighborhood then the clustering process starts and the current data point becomes the first point in a new cluster. Otherwise, the point will be labeled as noise (later this noisy point might become the part of any cluster). In both cases that point is marked as "visited".



3. For this first point in the new cluster, the points within its ε distance neighborhood also become part of the same cluster. This procedure of making all points in the ε neighborhood belong to the same cluster is then repeated for all of the new points that have been just added to the currently developing cluster.
4. The process is repeated until all points in the cluster are determined, i.e. all points within the ε neighborhood of the cluster have been visited and labeled as „visited".
5. Once we are done with the current cluster, a new unvisited point is retrieved and processed in the same way, creating another cluster or a noise point. This process repeats until all points are marked as visited and belong to a cluster or being a noise.

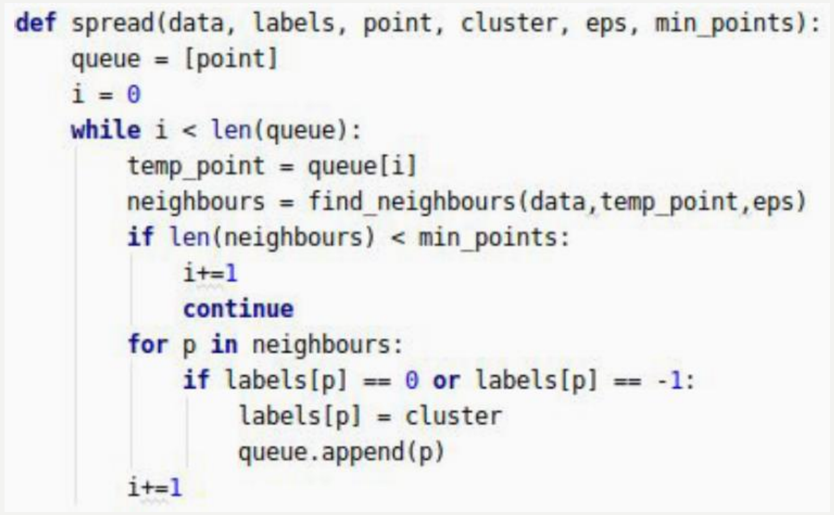
Example Implementation of DBSCAN

find_neighbours(data, point, eps) - finds all neighbours to the "point" with the radius "eps": Function uses a lambda filter.

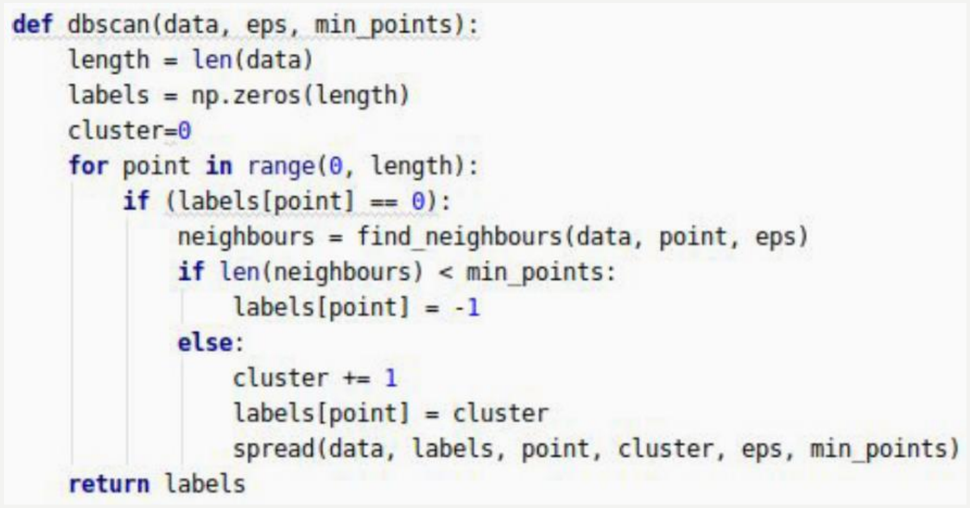


```python
def find_neighbours(data, point, eps):
    f = list(filter(lambda x: (np.linalg.norm(data[point] - data[x])<eps), range(len(data))))
    return f
```

dbscan(data, eps, min_points) - performs the full algorithm and returns all labels assigned to the nodes in the "data":



```python
def spread(data, labels, point, cluster, eps, min_points):
    queue = [point]
    i = 0
    while i < len(queue):
        temp_point = queue[i]
        neighbours = find_neighbours(data,temp_point,eps)
        if len(neighbours) < min_points:
            i+=1
            continue
        for p in neighbours:
            if labels[p] == 0 or labels[p] == -1:
                labels[p] = cluster
                queue.append(p)
        i+=1
```



```python
def dbscan(data, eps, min_points):
    length = len(data)
    labels = np.zeros(length)
    cluster=0
    for point in range(0, length):
        if (labels[point] == 0):
            neighbours = find_neighbours(data, point, eps)
            if len(neighbours) < min_points:
                labels[point] = -1
            else:
                cluster += 1
                labels[point] = cluster
                spread(data, labels, point, cluster, eps, min_points)
    return labels
```

spread(data, labels, point, cluster, eps, min_points) - grows the given cluster by finding all points belonging to it. This function is based on the BFS (breadth frist search) algorithm.

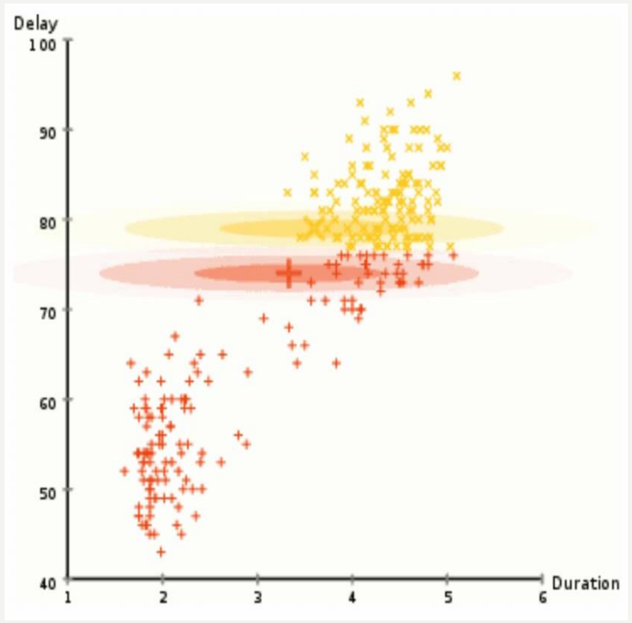
Expectation-Minimization Clustering

Expectation–Maximization (EM) Clustering using Gaussian Mixture Models (GMM)

- assumes that the data points are Gaussian distributed, i.e. we have two parameters to describe the shape of the clusters: the mean and the standard deviation. Therefore, the clusters can take any kind of elliptical shape, and each Gaussian distribution is assigned to a single cluster.
- In order to find the parameters of the Gaussian distribution for each cluster (i.e. the appropriate mean and standard deviation), we use an Expectation–Maximization (EM) optimization algorithm.

There are two key advantages to using GMMs:

- GMMs (ellipses) are a lot more flexible in terms of cluster covariance than K-Means (circles) due to the standard deviation parameter. K-Means is actually a special case of GMMs in which each cluster's covariance along all dimensions approaches 0.
- Since GMMs use probabilities, they can have multiple clusters per data point. So if a data point is in the middle of two overlapping clusters, we can simply define its class by the probability of belonging to class 1 and another probability of belonging to class 2, i.e. GMMs support mixed membership.

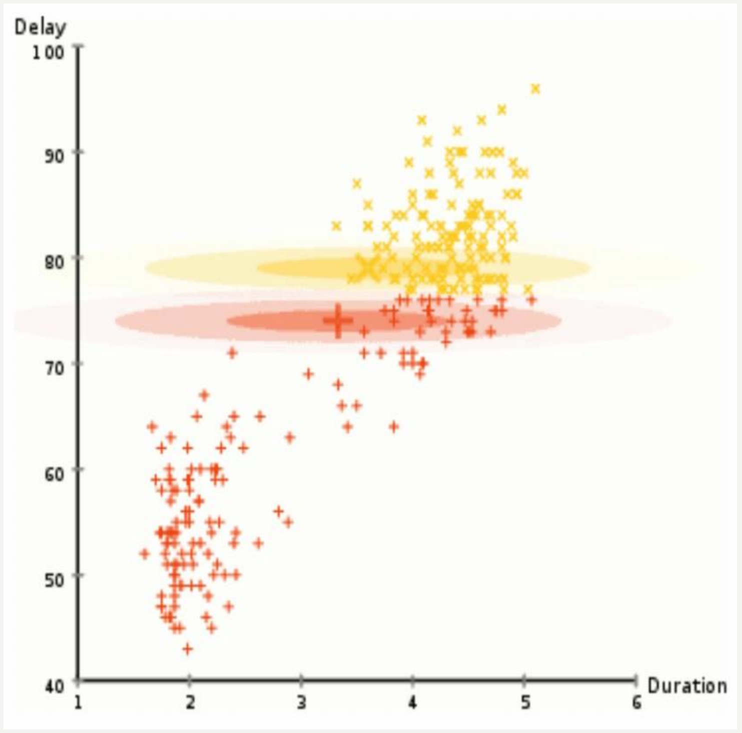


The disadvantage is that EM Clustering requires preset the number of clusters.

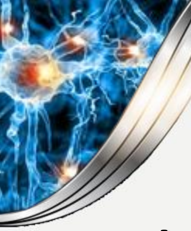
Expectation-Minimization Clustering

Expectation–Maximization Algorithm

1. Select a number of clusters and randomly initialize the Gaussian distribution parameters for each cluster.
2. For the given Gaussian distributions for each cluster, compute the probabilities of belonging of data points to particular clusters. The closer a point is to the Gaussian's center, the more likely it belongs to that cluster. This should make sense since with a Gaussian distribution, we assume that most of the data lie closer to the center of the cluster.
3. Based on these probabilities, we compute a new set of parameters for the Gaussian distributions such that we maximize the probabilities of data points within the clusters. We compute these new parameters using a weighted sum of the data point positions, where the weights are the probabilities of the data point belonging in that particular cluster.
4. The above steps are iteratively repeated until convergence, where the distributions do not change much from iteration to iteration.

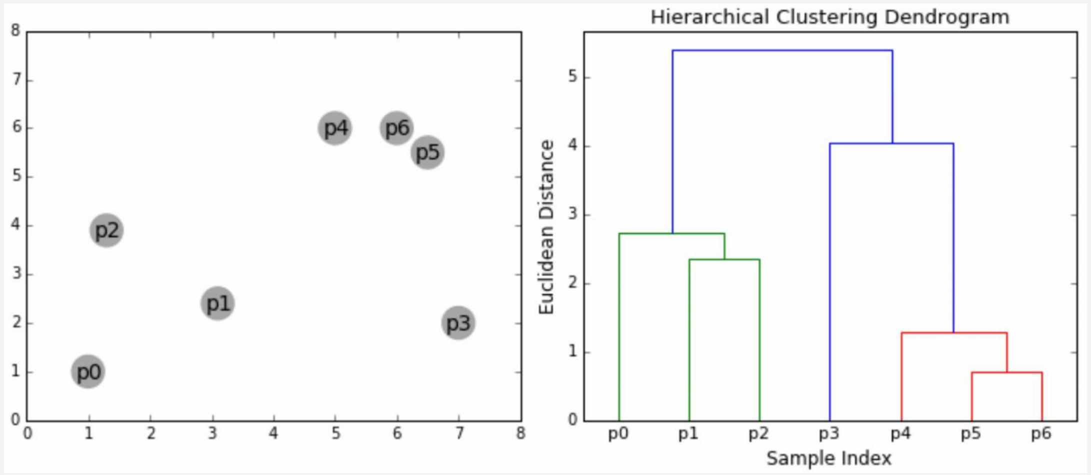


Agglomerative Hierarchical Clustering



Agglomerative Hierarchical Clustering

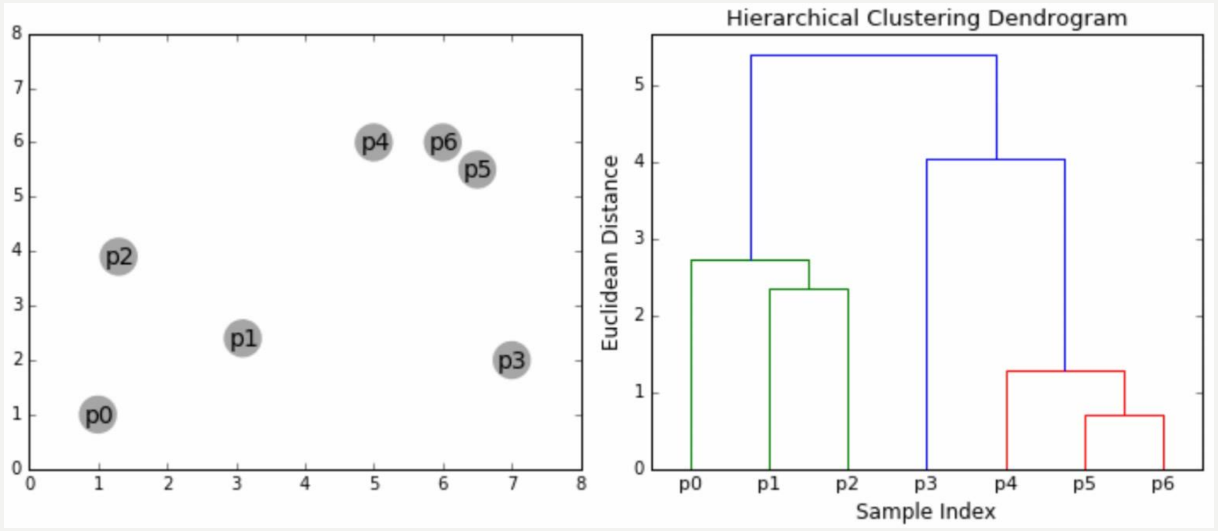
- Is a bottom-up algorithm which treats each data point as a single cluster at the outset and then it successively merges (agglomerate) pairs of clusters until all clusters have been merged into a single cluster that contains all data points. This hierarchy of clusters is represented as a tree (dendrogram). The root of the tree is the unique cluster that gathers all the samples, the leaves being the clusters with only one sample.
- Hierarchical clustering does not require to specify the number of clusters, and we can even select which number of clusters looks best after we have built the tree. The algorithm is not sensitive to the choice of distance metric; all of them tend to work well whereas the choice of distance metric is critical for other clustering algorithms.
- A particularly good use case of hierarchical clustering methods is when the underlying data has a hierarchical structure and you want to recover this hierarchy. The cost of hierarchical clustering has time complexity of $O(N^3)$, unlike the linear complexity $O(N)$ of K-Means and GMM.



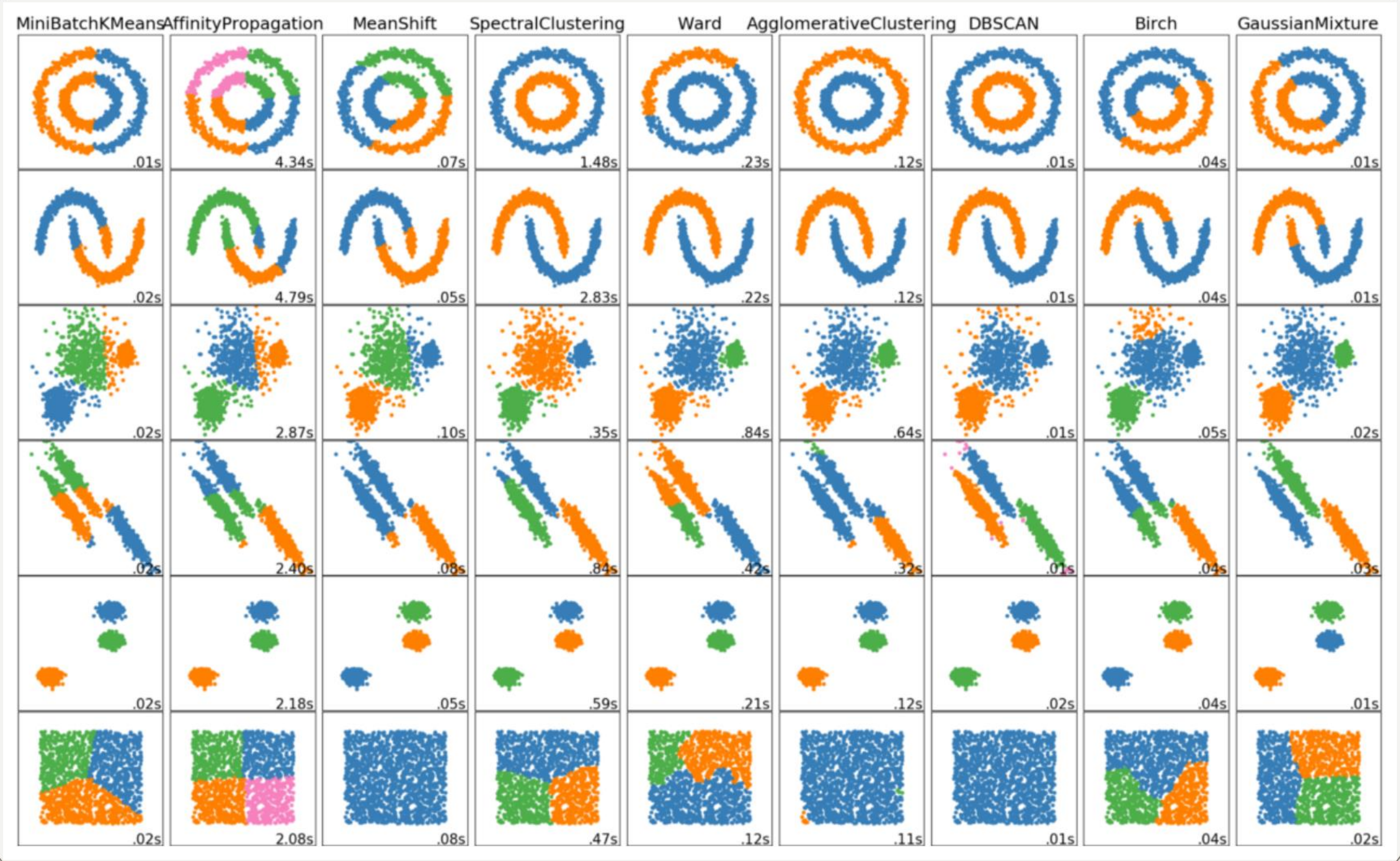
Agglomerative Hierarchical Clustering

Agglomerative Hierarchical Clustering Algorithm:

1. At the beginning, treat each data point as a single cluster, i.e. if there are N data points in our dataset then we have N clusters. Then select a distance metric that measures the distance between two clusters. Use an average linkage which defines the distance between two clusters to be the average distance between data points in the first cluster and data points in the second cluster.
2. On each iteration, combine two clusters into one when the clusters have the smallest average linkage, i.e. according to our selected distance metric, these two clusters have the smallest distance between each other and therefore are the most similar and should be combined.
3. Repeat step 2 until the root of the tree is reached, i.e we achieved one cluster that contains all data points. In this way, we can select how many clusters we want to have in the end, simply by choosing when to stop combining the clusters, i.e. when we stop building the tree.



Comparison of Clustering Algorithms



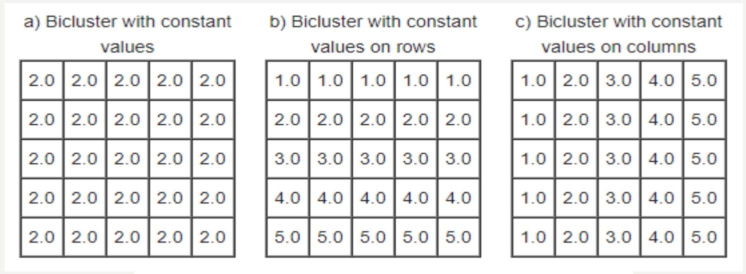
We can notice the pros and cons of the different clustering algorithms on difficult data points.

Biclustering and Triclustering

Biclustering

- called also block clustering, co-clustering, or two-mode clustering is a data mining technique that allows simultaneous clustering of the rows and columns of a matrix;
- is searching for the similarities of these points in a space of input data taking into account only a subset of attributes (e.g. rows);
- does not require input data to be labeled as in the classification tasks;
- can group data into various groups, and next, these groups can be labeled;
- use unsupervised learning to adapt a model.

Definition: Given a set of *m* samples represented by an n-dimensional feature vector, the entire dataset can be represented as m rows in n columns (i.e., an m × n matrix).
The biclustering algorithm generates biclusters, i.e., a subset of rows which exhibit similar behavior across a subset of columns, or vice versa.



a) Bicluster with constant values

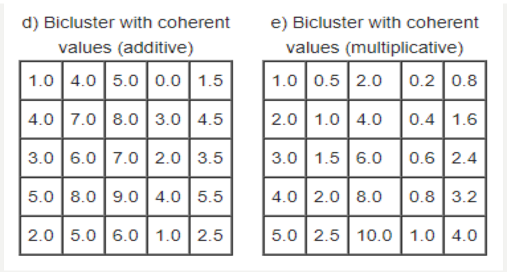
2.0	2.0	2.0	2.0	2.0
2.0	2.0	2.0	2.0	2.0
2.0	2.0	2.0	2.0	2.0
2.0	2.0	2.0	2.0	2.0
2.0	2.0	2.0	2.0	2.0

b) Bicluster with constant values on rows

1.0	1.0	1.0	1.0	1.0
2.0	2.0	2.0	2.0	2.0
3.0	3.0	3.0	3.0	3.0
4.0	4.0	4.0	4.0	4.0
5.0	5.0	5.0	5.0	5.0

c) Bicluster with constant values on columns

1.0	2.0	3.0	4.0	5.0
1.0	2.0	3.0	4.0	5.0
1.0	2.0	3.0	4.0	5.0
1.0	2.0	3.0	4.0	5.0
1.0	2.0	3.0	4.0	5.0



d) Bicluster with coherent values (additive)

1.0	4.0	5.0	0.0	1.5
4.0	7.0	8.0	3.0	4.5
3.0	6.0	7.0	2.0	3.5
5.0	8.0	9.0	4.0	5.5
2.0	5.0	6.0	1.0	2.5

e) Bicluster with coherent values (multiplicative)

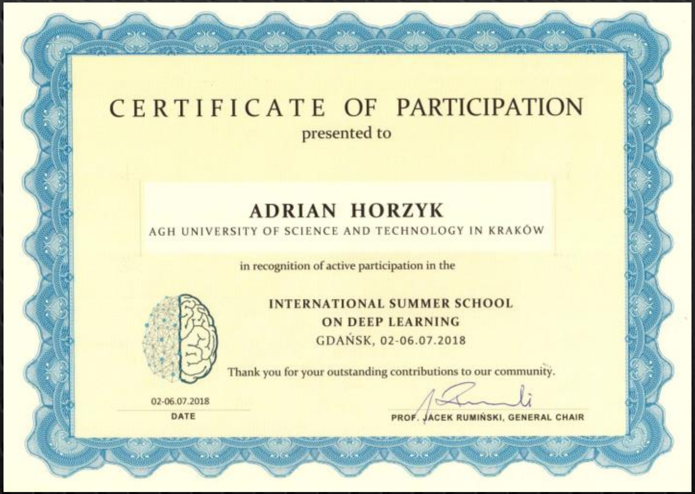
1.0	0.5	2.0	0.2	0.8
2.0	1.0	4.0	0.4	1.6
3.0	1.5	6.0	0.6	2.4
4.0	2.0	8.0	0.8	3.2
5.0	2.5	10.0	1.0	4.0

Triclustering

- additionally takes into account the time and searches for patterns that frequently repeat in time. Clusters of such frequently repeated patterns spanned over time will be the result of triclustering.

BIBLIOGRAPY

1. Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, MIT Press, 2016, ISBN 978-1-59327-741-3 or PWN 2018.
2. Holk Cruse, Neural Networks as Cybernetic Systems, 2nd and revised edition
3. https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68
4. https://en.wikipedia.org/wiki/Biclustering
5. Home page for this course: http://home.agh.edu.pl/~horzyk/lectures/ahdydci.php



CERTIFICATE OF PARTICIPATION

presented to

ADRIAN HORZYK

AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY IN KRAKÓW

in recognition of active participation in the

INTERNATIONAL SUMMER SCHOOL ON DEEP LEARNING

GDAŃSK, 02-06.07.2018

Thank you for your outstanding contributions to our community.

02-06.07.2018

DATE

[illegible]

PROF. JACEK RUMIŃSKI, GENERAL CHAIR