



UNIVERSITAT DE  
BARCELONA

# *Introduction to Machine Learning*

Master in Artificial Intelligence  
UPC, UB, URV





## Week 10

# Course. Introduction to Machine Learning Theory 10. Visualization

Dr. Maria Salamó Llorente  
[maria.salamo@ub.edu](mailto:maria.salamo@ub.edu)

Dept. Mathematics and Informatics,  
Faculty of Mathematics and Informatics,  
University of Barcelona (UB)



# Introduction to Machine Learning

## Supervised Learning

Non Linear Decision

Lazy Learning  
(K-NN, IBL,  
CBR)

Overfitting,  
model selection

Feature  
selection

Kernel  
Learning

Linear  
Decision

Perceptron,  
SVM

## Decision Learning Theory

Basic concepts  
of  
Decision  
Learning  
Theory

Bias/Variance,  
VC dimension,  
Practical advice  
of how to use  
learning  
algorithms

## Unsupervised Learning

Cluster  
Analysis

K-Means,  
Fuzzy C-  
means,  
EM

Factor  
Analysis

PCA, ICA

## Visualization

**Self  
Organized  
Maps (SOM)  
, Multi-  
Dimensional  
Scaling**

Applications  
of ML

RecSys

Beyond ML

Bias and  
Fairness in ML

## Machine Learning in practice

1

## ***Introduction to Visualization***

2

## ***Self-Organizing Maps (SOM)***

3

## ***Multi-dimensional Scaling (MDS)***



# 1. Introduction to Visualization



- **Objective:**

Visualizing low-dimensional views of high-dimensional data

- This lecture is mainly devoted to:

- Self-Organizing Maps
- Multi-Dimensional Scaling



Consider the following ideas

knowledge

data

wisdom

information

- Now, order previous terms as logically as possible

\_\_\_\_\_ > \_\_\_\_\_ > \_\_\_\_\_ > \_\_\_\_\_



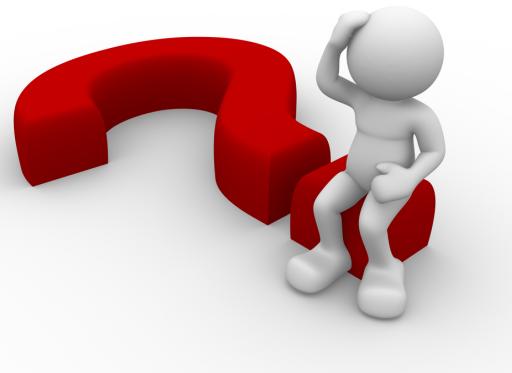
***Take a moment to think about it***

....

***make a note that explain your decision***  
(You will have the answer in the online session)

# Connect keywords

*Connect each keyword  
with its corresponding  
image*



data



information



knowledge



wisdom



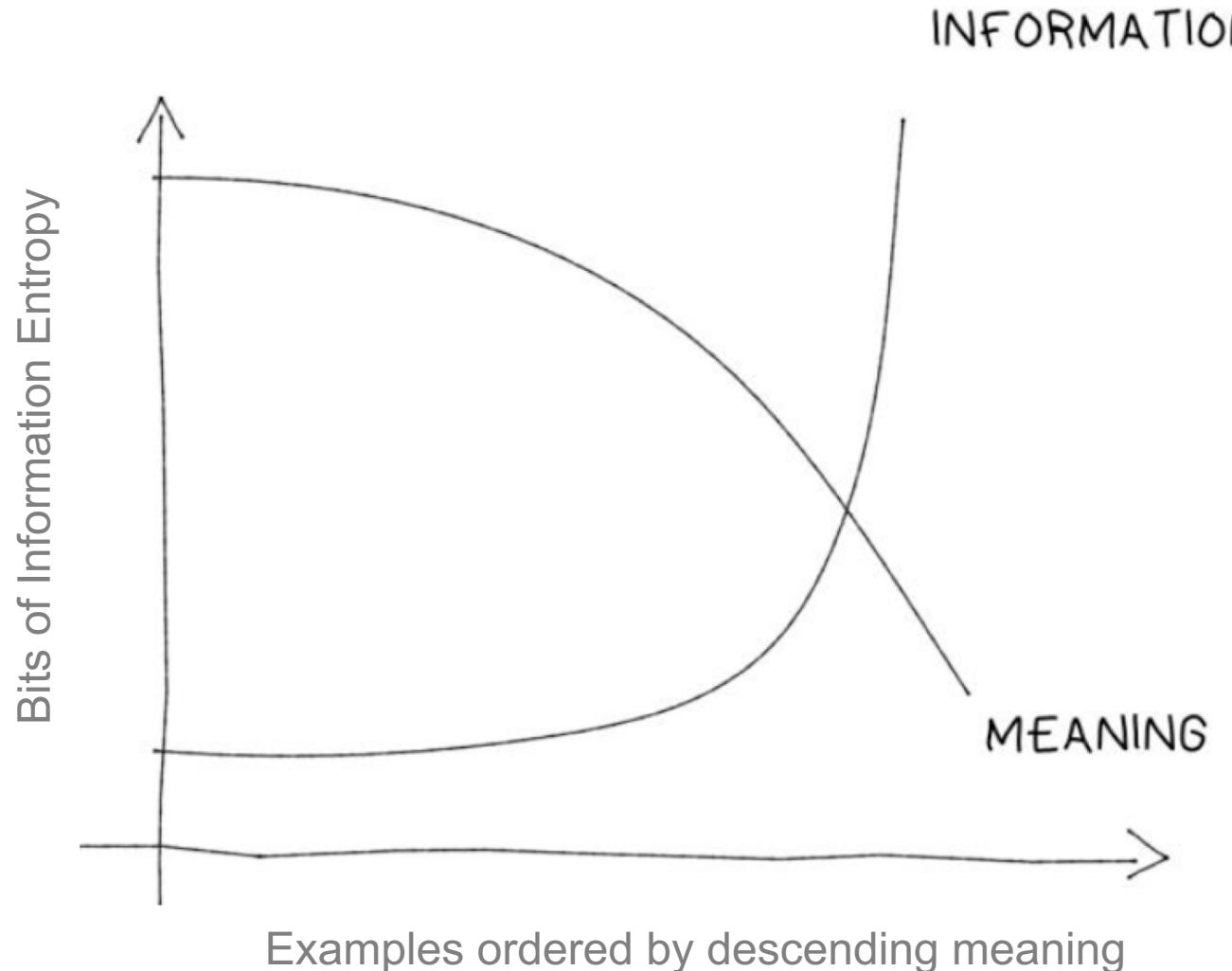


# How is wisdom made?



*One way to arrive at wisdom is the **Synthesis** process*

# Information vs Meaning



*There is no peace in more*

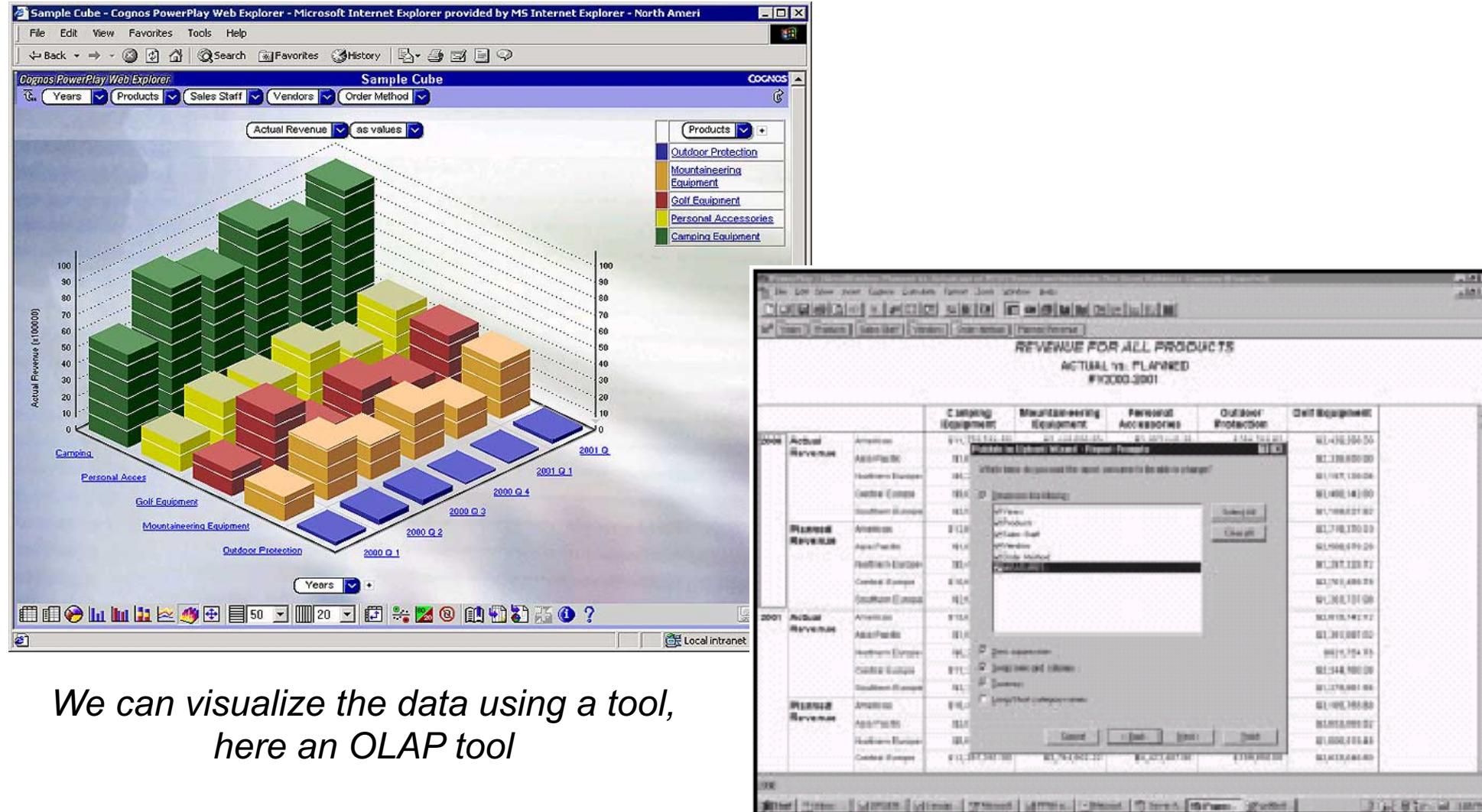
**“Less (info) is more (meaning)”**



**VISUALIZATION** is the study of the visual representation of the data

**Main goal** is to communicate the information clearly and effectively through graphical means

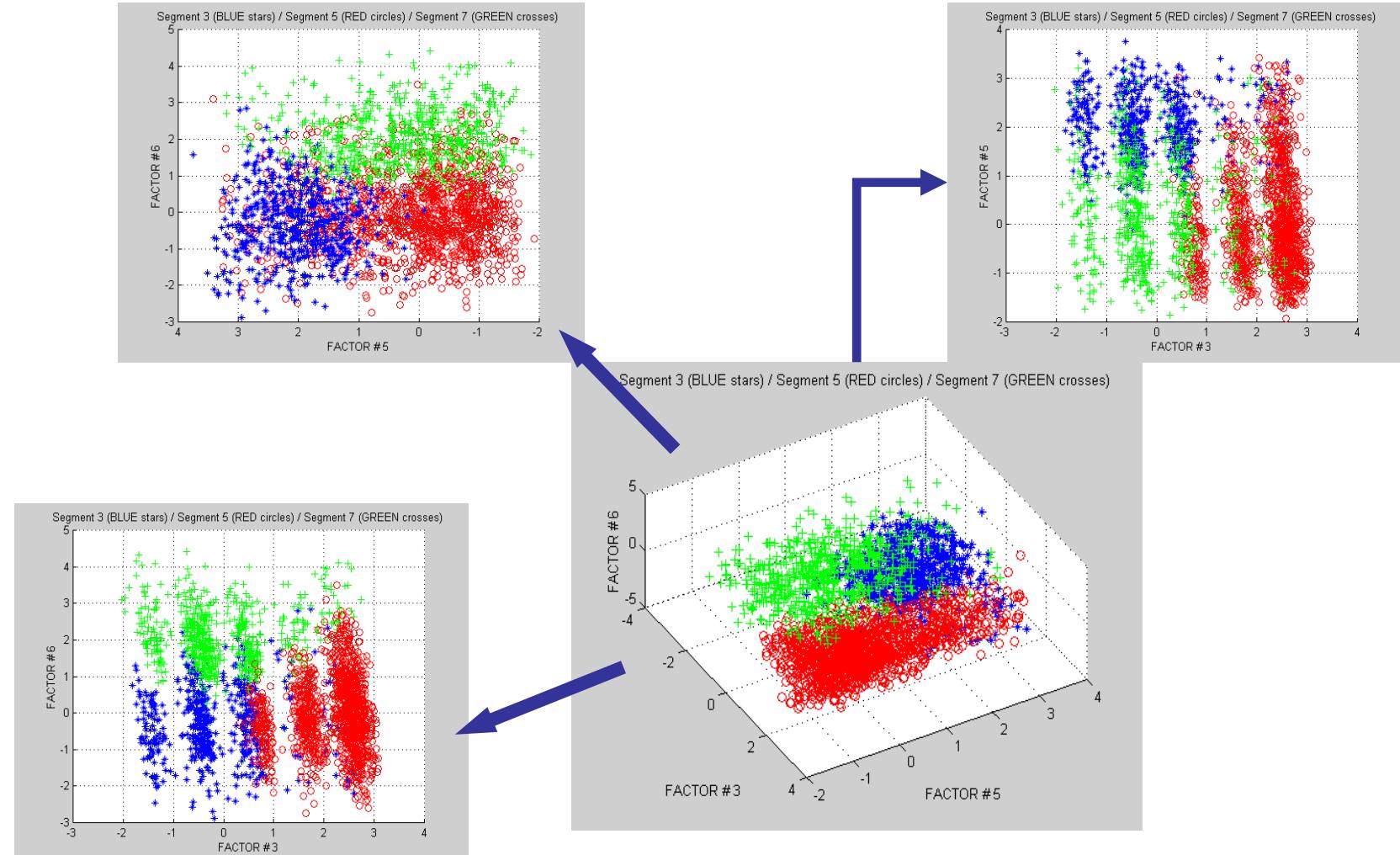
# Data visualization vs model visualization



We can visualize the data using a tool,  
here an OLAP tool

# Data visualization vs model visualization

... or we can visualize a **model**. Here we visualize in 3D three factors of interest in a factor analysis solution

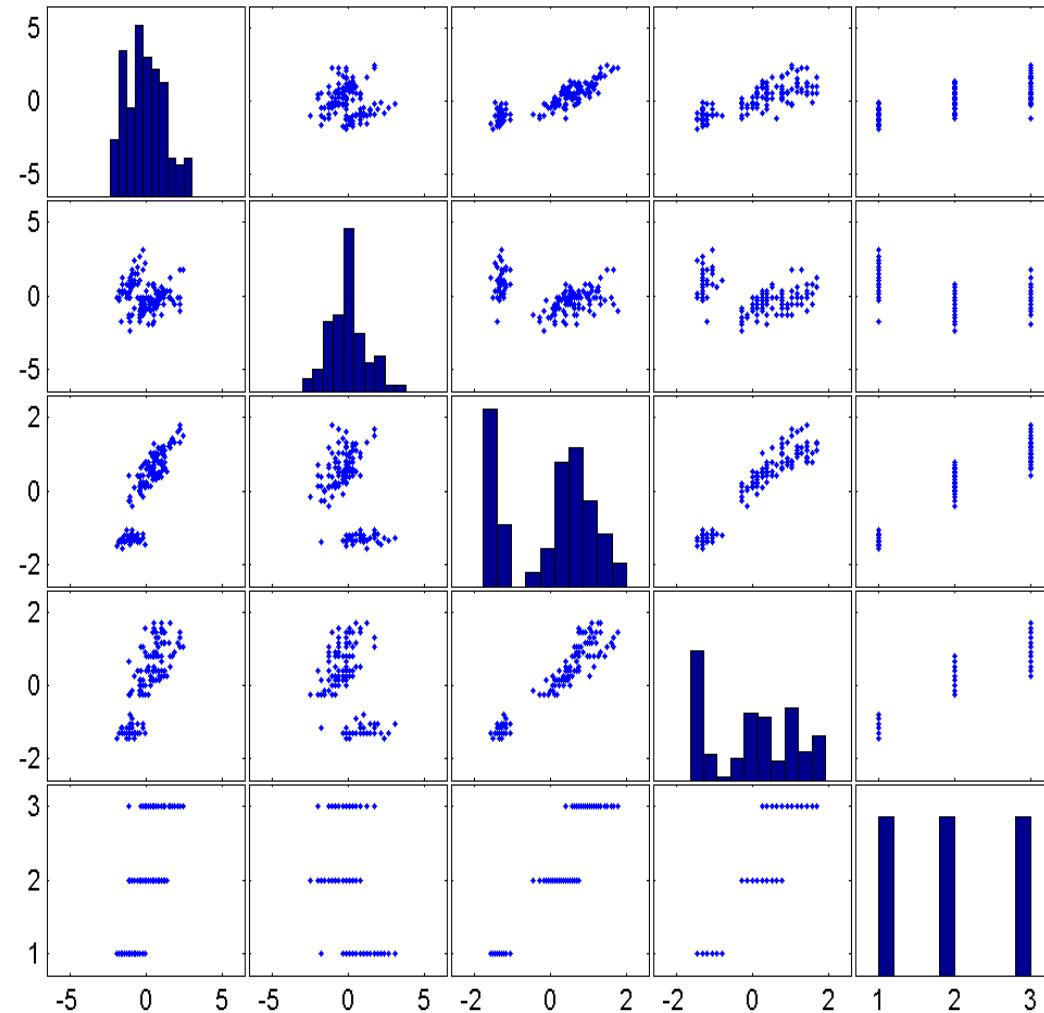




## How do we visualize data of high (or even very high) dimensionality?

- **Eliminate dimensions** (data variables): those which are redundant and/or uninformative (at least you manage to alleviate part of the problem...)
- **Divide & conquer**: a classic approach where you create multiple visualizations of low dimensionality
- **Latent and projection models**

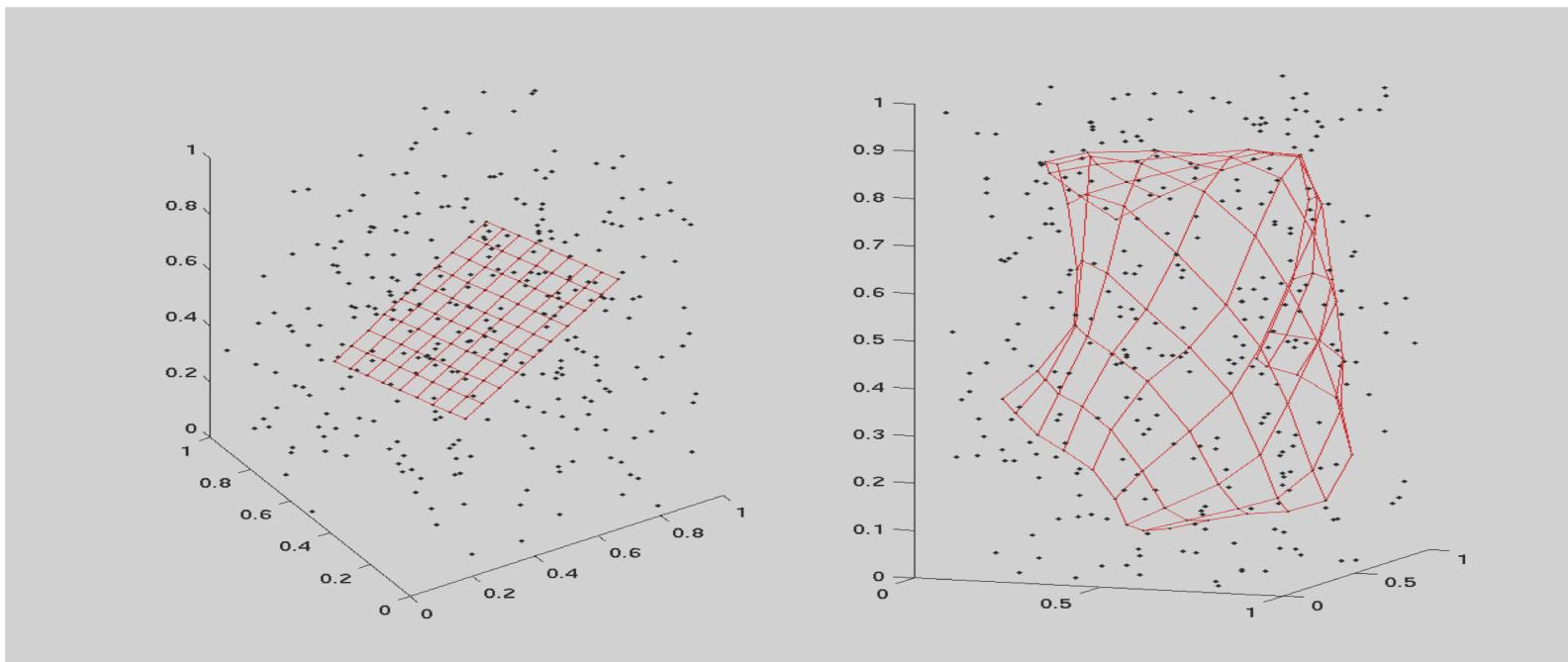
This is a divide & conquer approach



- **Projection**
  - Dimensionality compression
    - **Multi-dimensional Scaling**
  - Similitude information coding
- **Clustering**
  - Finding grouping structure in data
    - E.g., **Principal Components Analysis**
  - Similitude information coding
- ***Self-Organizing Map (SOM) & Generative Topographic Mapping (GTM)***
  - They combine latent representation and clustering

# Projection

- Representation in <4-D, so that the distance-neighborhood relations between multi-dimensional points are faithfully preserved
  - It is impossible to preserve information integrally
  - Some scale normalization is required
- Linear vs. non-linear projections



# Projection methods

- **Methods based on inter-point distances, where:**

$d_x$  = distance in the original space

$d_y$  = distance in the projection space

$h$  = neighborhood function

$$E = \sum (d_x - d_y)^2$$

**MDS, PCA**

$$E = \sum (d_x - d_y)^2 / d_x$$

**Sammon's projection**

$$E = \sum (d_x - d_y)^2 e^{-d_y}$$

**CCA**

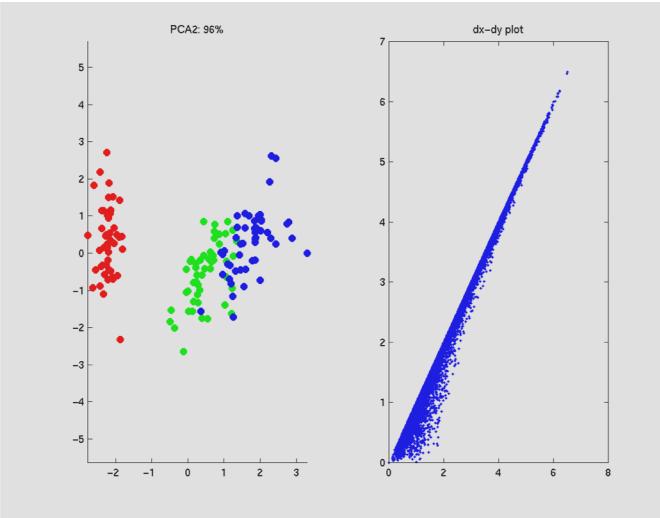
$$E = \sum d_x^2 h(d_y)$$

**SOM**

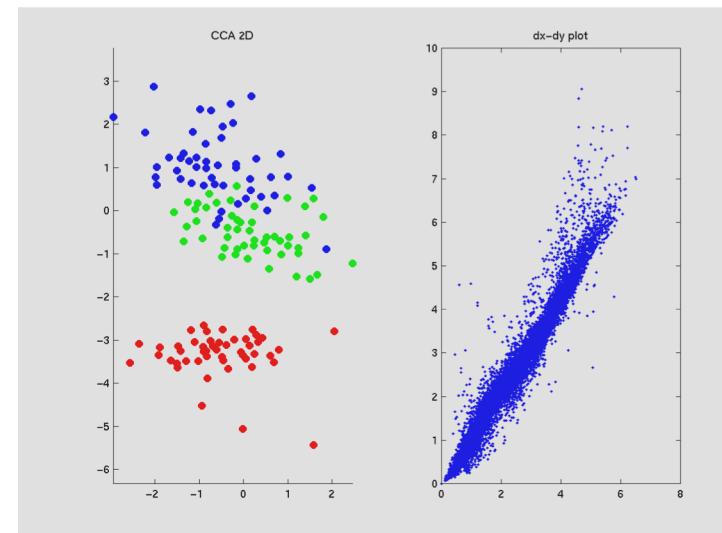
... and in which we aim to minimize an inherent projection distortion ( $E$ )

# Example of Projection

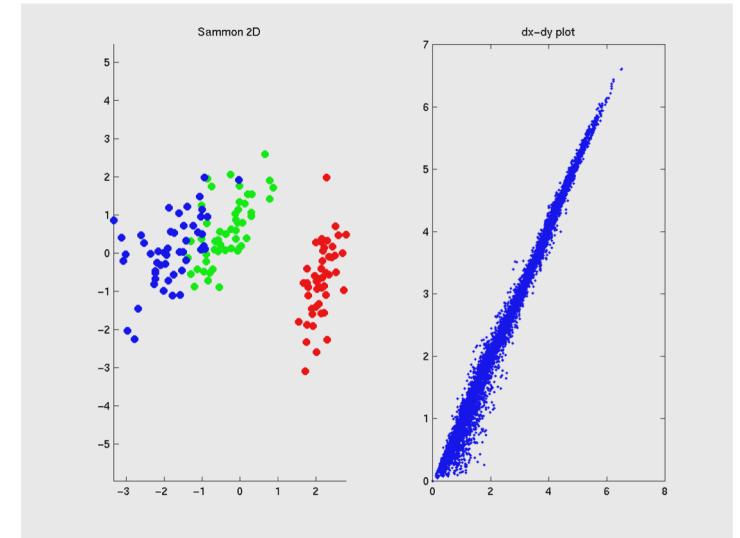
PCA



CCA



Sammon's projection



## 2. Self-Organizing Maps (SOM)



*Look at the videos*

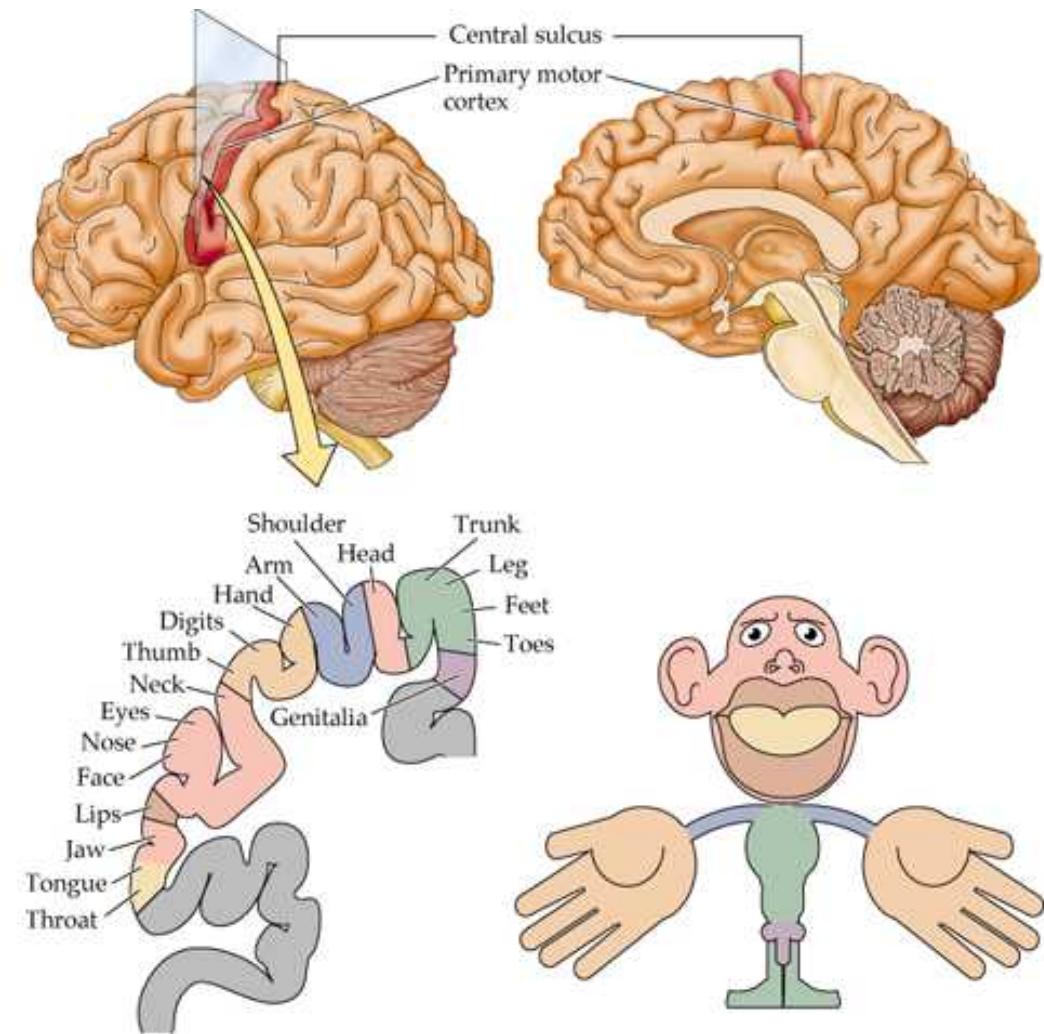
<https://youtu.be/K4WuE7zIOZo>

[https://youtu.be/0qtvb\\_Nx2tA](https://youtu.be/0qtvb_Nx2tA)



1. Introduction to visualization
2. Self-Organizing Maps
  1. Introduction to Self-Organizing Maps
  2. Defining SOM : Output layers, topological structure, neurons, the adaptation in SOM, training process
  3. SOM algorithm
  4. Example
  5. Summary
3. Multi-dimensional Scaling
  1. Introduction to multidimensional scaling
  2. Taxonomy of MDS
  3. Classical MDS algorithm
  4. Example
  5. Summary

- T. Kohonen proposes SOM in 1982
- **SOM are simplified models of cortical maps in the brain**
- Things that are near in the outside world link to areas near in the cortex
- These areas are represented by two-dimensional maps
- Neighboring areas in these maps represent neighboring areas in the sensory input space

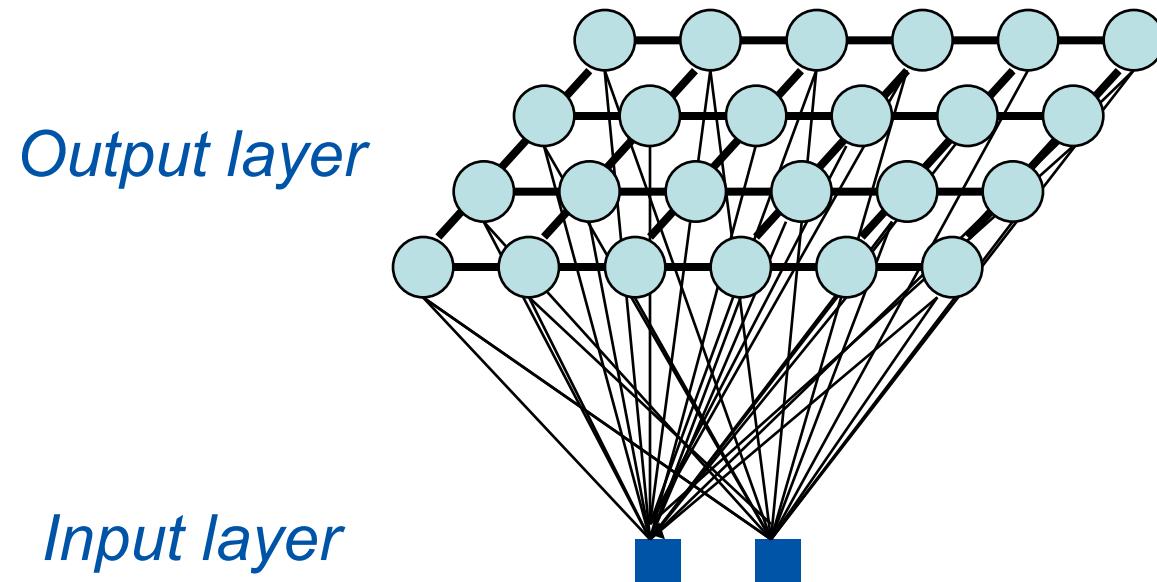




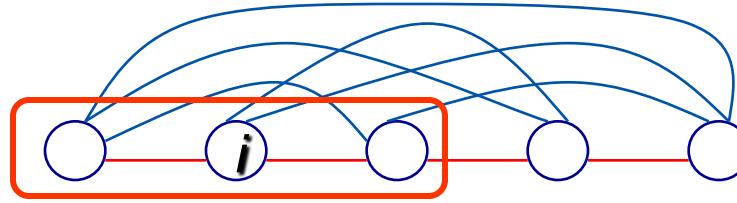
- Kohonen describes SOM as “**visualization and analysis tool for high dimensional data**”
- It is an unsupervised ANN
- Belongs to the category of competitive learning networks
- Applications
  - Clustering
  - Dimensionality reduction
  - Classification
  - Feature extraction (Self-Organized Feature Map)
  - Visualization

# Self organizing maps

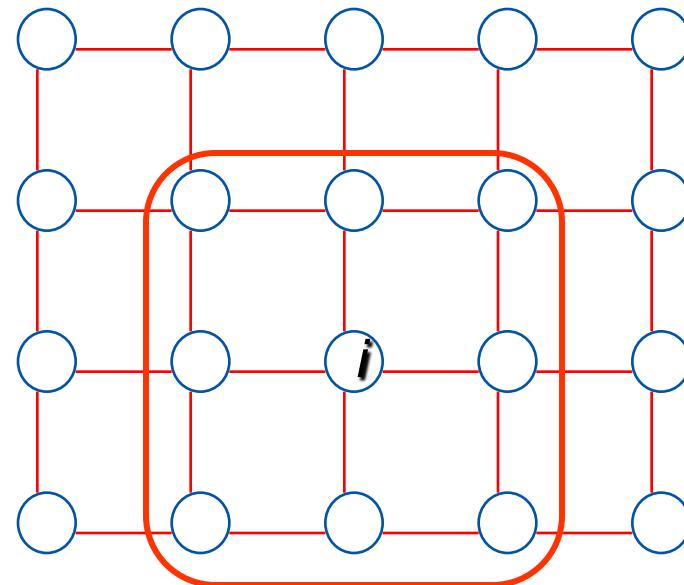
- The purpose of SOM is to map a multi-dimensional input space onto a **topology preserving** map of neurons
  - Preserve a topological so that neighboring neurons respond to «similar» input patterns
  - The topological structure is often a 2 or 3 dimensional space



# Commonly output-layer structures



**One-dimensional**  
(completely interconnected  
for determining “winner” unit)

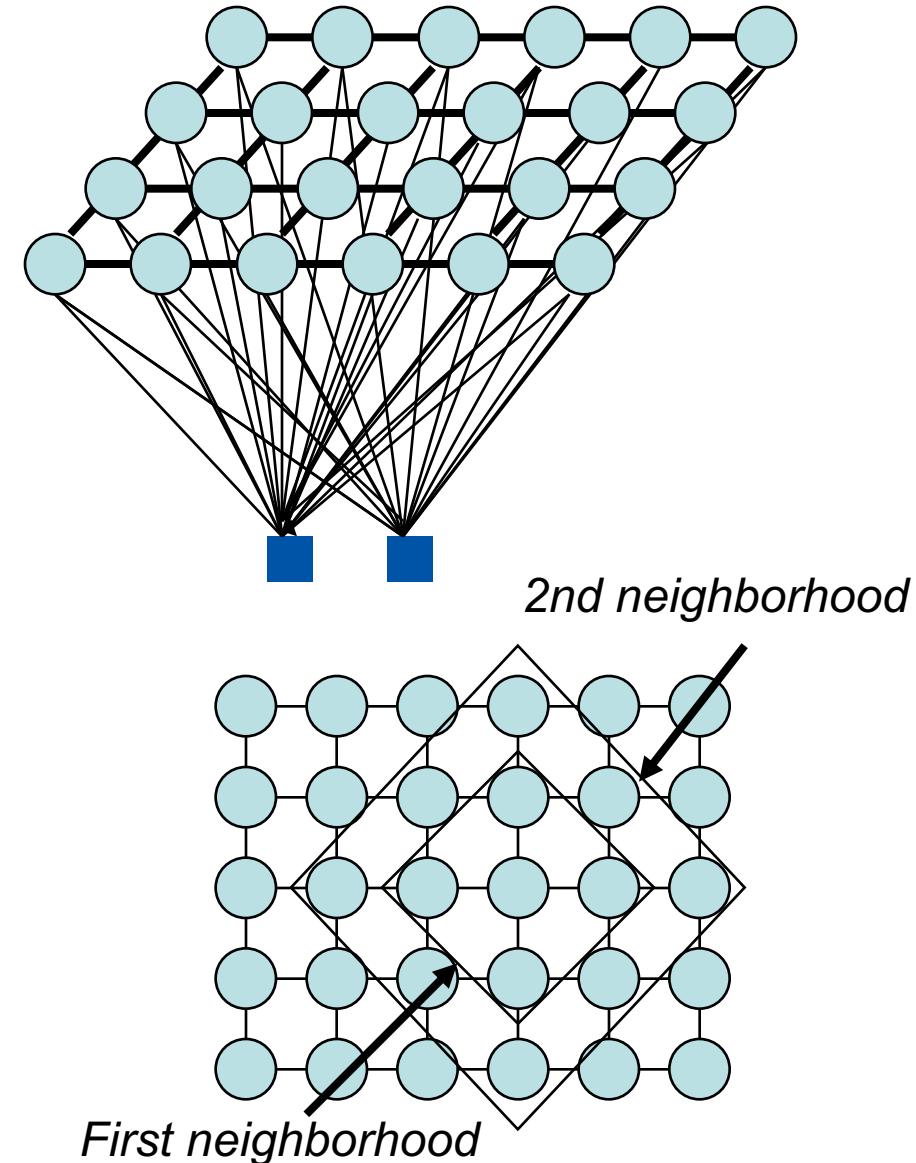


**Two-dimensional**  
(connections omitted, only neighborhood  
relations shown [green])

Neighborhood of neuron i

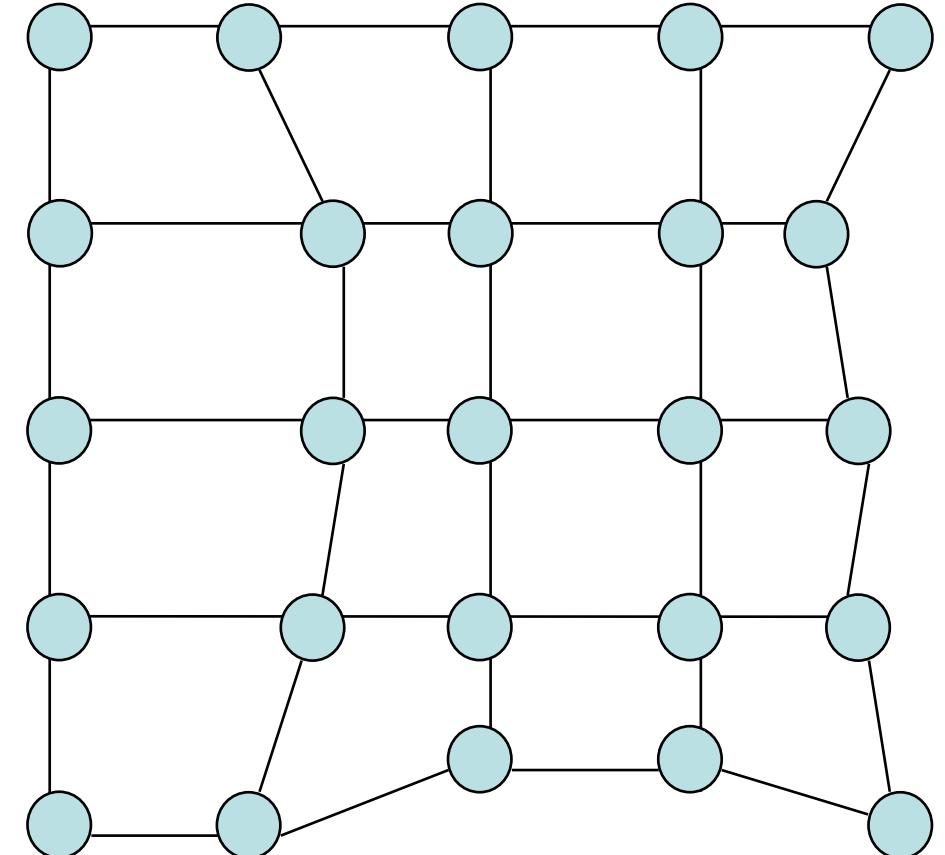
# Topological structure

- The activation of the neuron is spread in its direct neighborhood
  - neighbors become sensitive to the same input patterns
- Block distance
- The size of the neighborhood is initially large but reduce over time
  - Specialization of the network



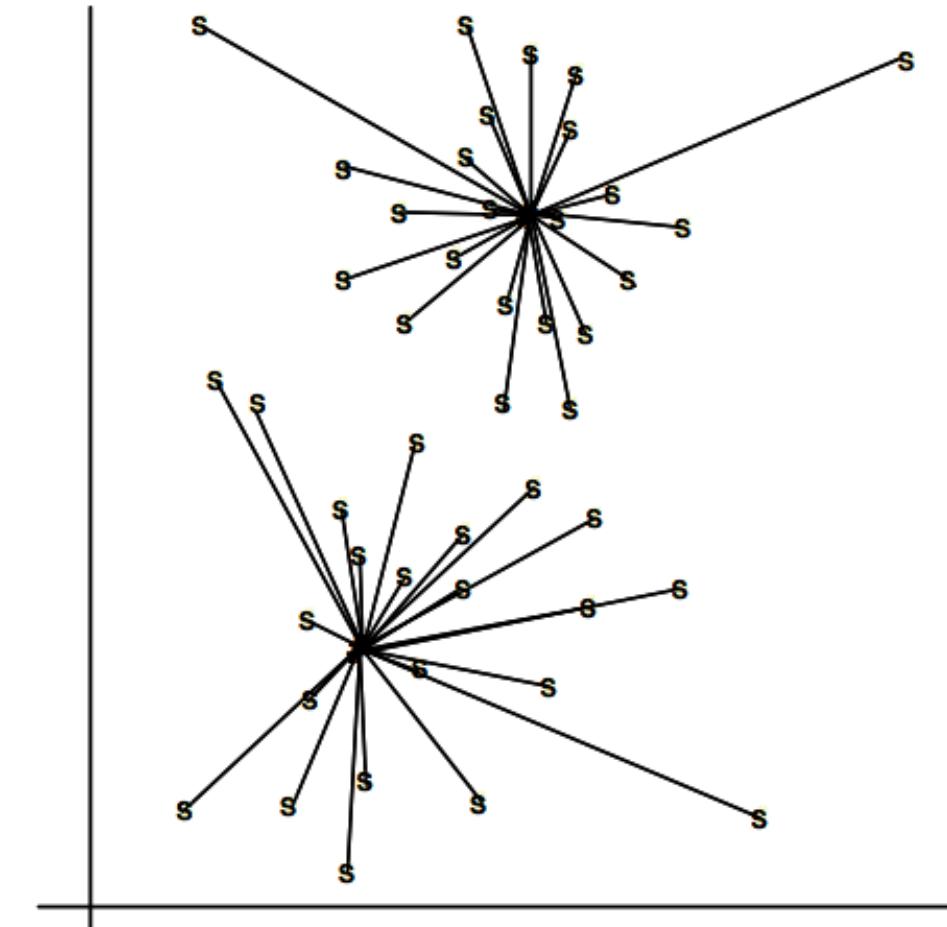
- SOM maps a multi-dimensional input space onto a topology preserving **map of neurons**
- Each neuron is assigned a weight vector with the same dimensionality of the input space
- Each Input pattern is compared to each weight vector
  - Distance is calculated between the input pattern and each neuron in the network (e.g. *Euclidean Distance*)
- Selects the neuron that is closest as the winning neuron

- During **training**, the “winner” neuron and its neighborhood adapts to make their weight vector more similar to the input pattern that caused the activation
- The neurons are moved closer to the input pattern
- The magnitude of the adaptation is controlled via a **learning parameter** which decays over time



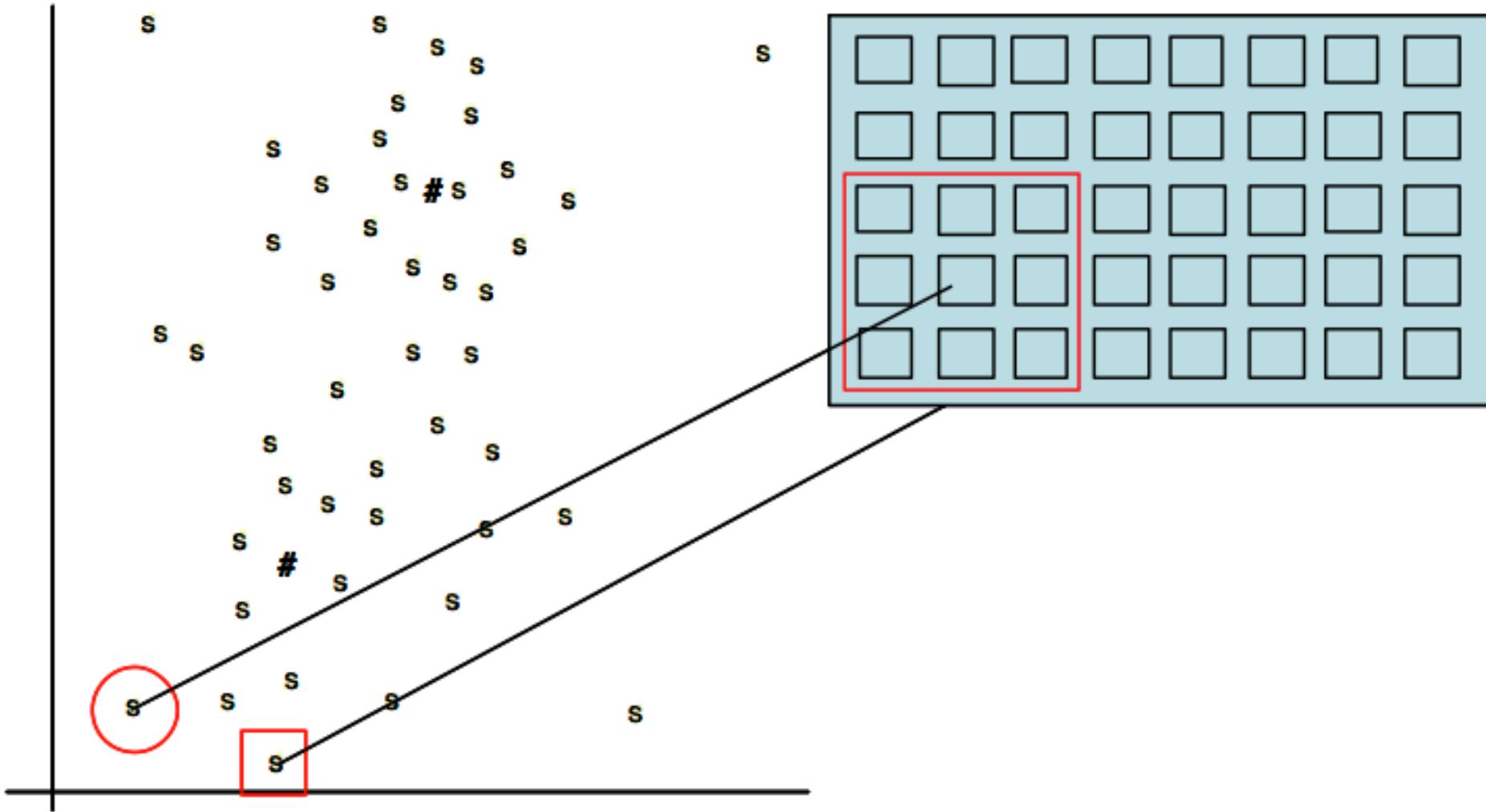
# Quantization error

- Measures how well our neurons represent the input patterns
  - Note that there will be always some difference between the input pattern and the neuron it is mapped to
- It is calculated by summing all the distances between each input pattern and the neuron to which is mapped.



- Evaluates the complexity of the output space
- Measures the number of times the second closest neighbor in the input space is not mapped into the neighbourhood of the neuron in the output space
- A high topological error may indicate that the classification problem is complex or may suggest that the training was not adequate and the network is folded

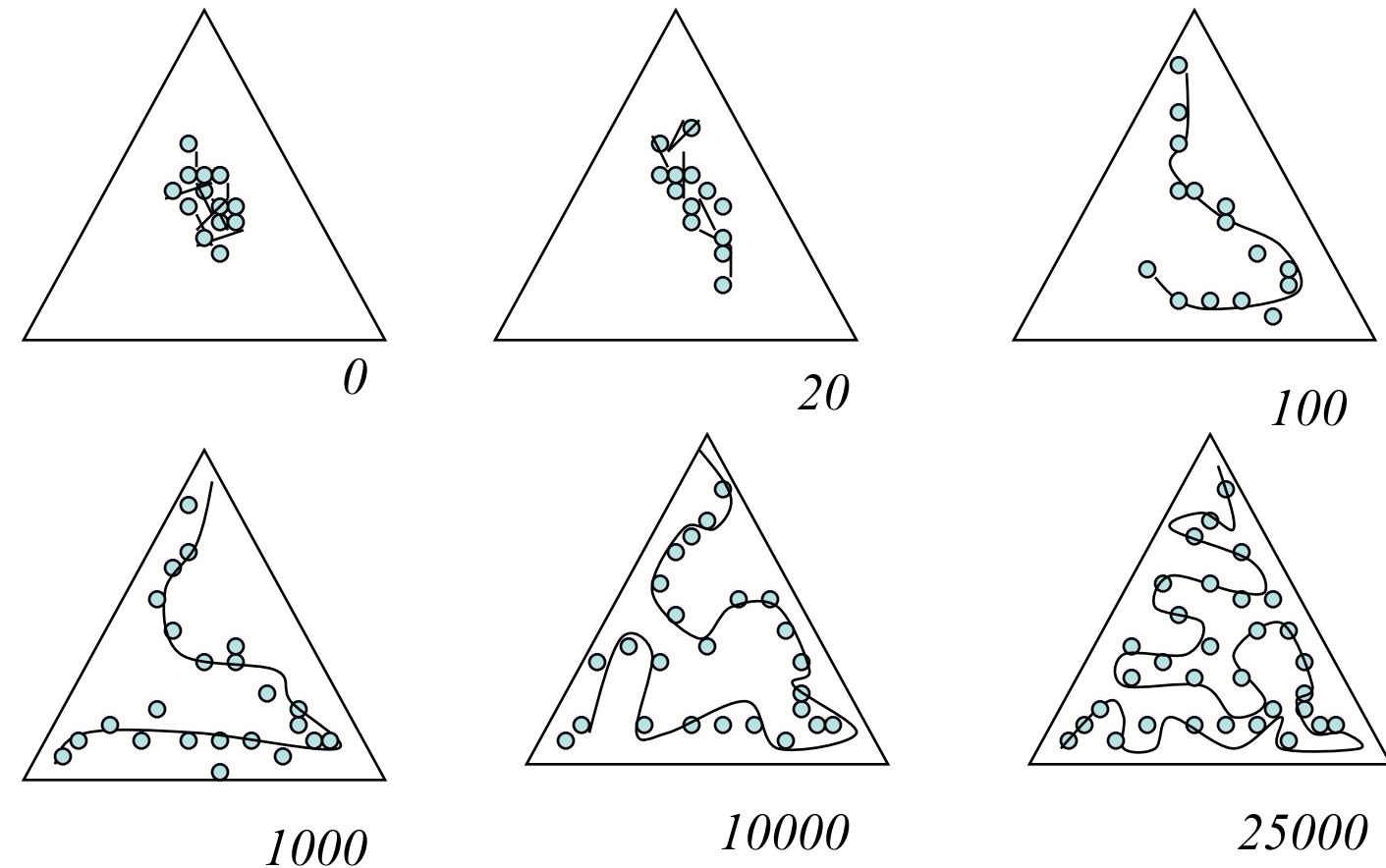
# Topological error



- Before training, neurons may be initialized randomly
- **Part 1. Unfolding phase**
  - Neurons are “spread out” and pulled towards the general area (in the input space) where they will stay
- **Part 2. Fine tuning phase**
  - SOM match the neurons as far as possible to the input patterns, thus decreasing the quantization error

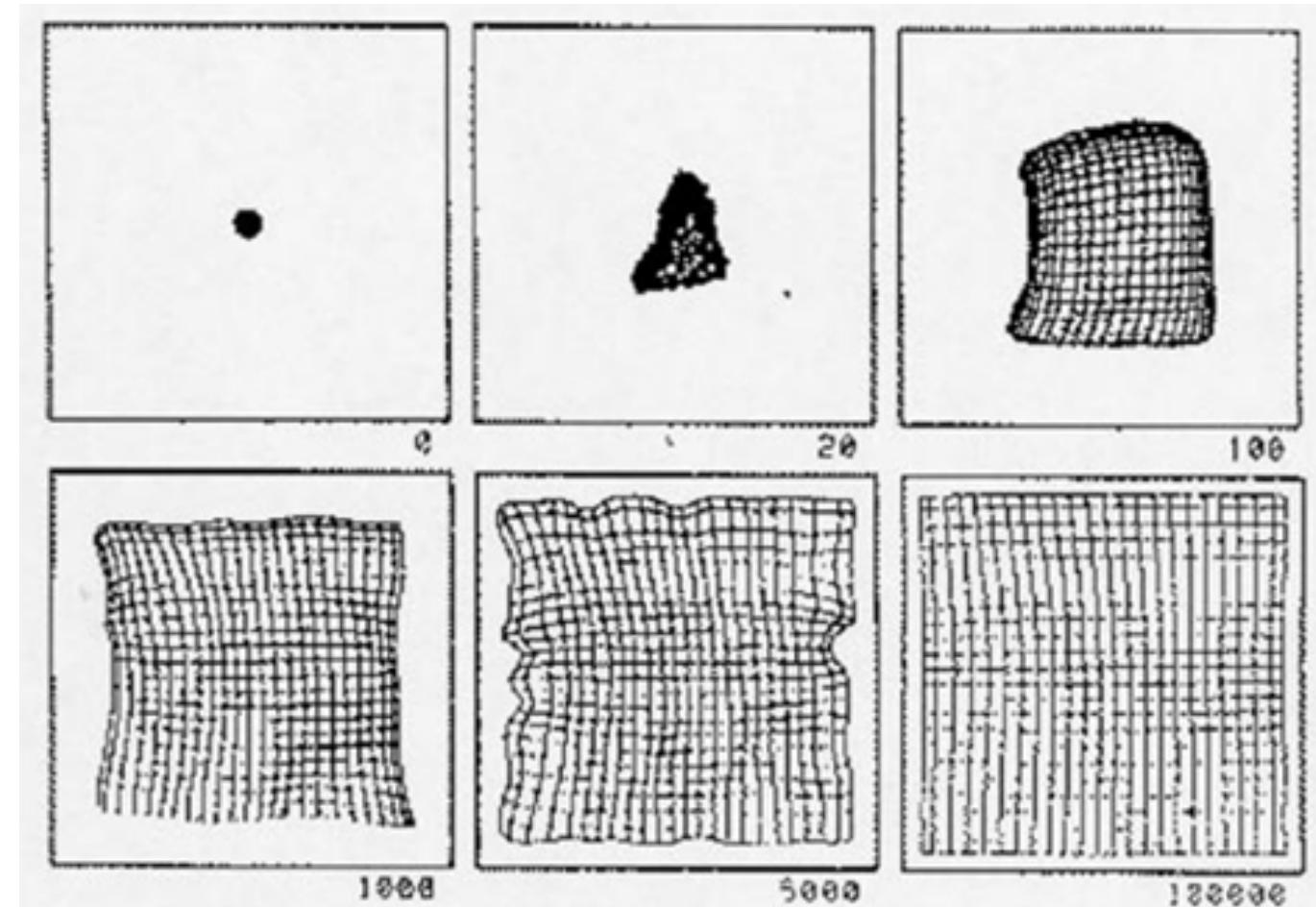
# Example 1 when training SOM

Learning a one-dimensional representation of a two-dimensional (triangular) input space:



# Example 2 when training a SOM

*Learning a two-dimensional representation of a two-dimensional (square) input space:*



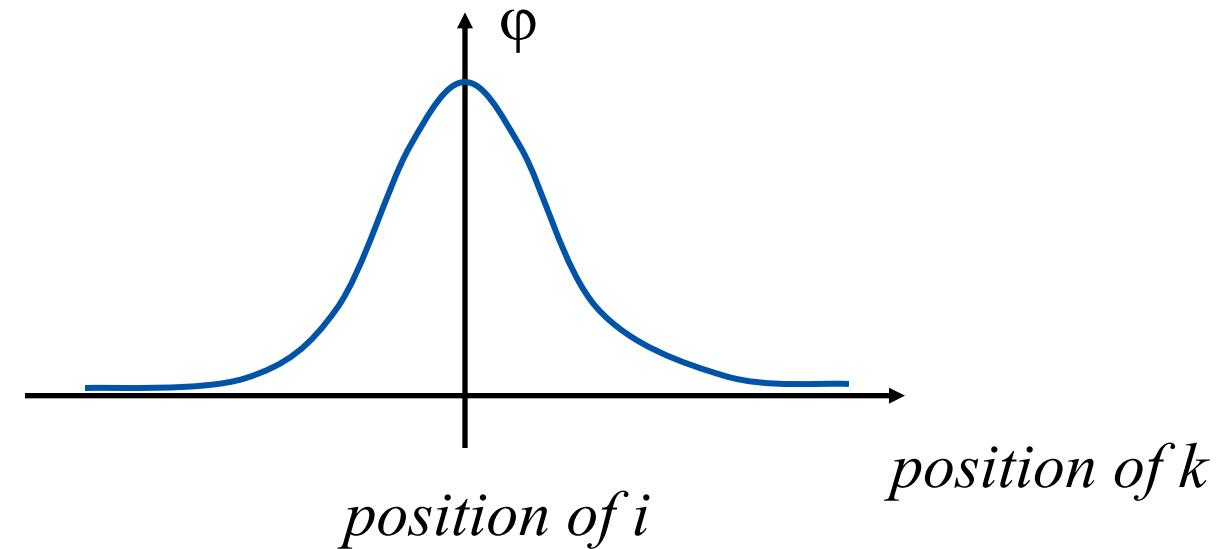
For  $n$ -dimensional input space and  $m$  output neurons:

1. Choose random weight vector  $w_i$  for neuron  $i$ ,  $i = 1, \dots, m$
2. For each input, choose random input  $x$  and do 2, 3, 4, and 5
3. Calculate the distance between the pattern and all neurons  
**(usually the Euclidean distance)**
4. Determine winner neuron  $k$ :  $\|w_k - x\| = \min_i \|w_i - x\|$
5. Update all weight vectors of all neurons  $i$  in the neighborhood of neuron  $k$ :  
 $w_i := w_i + \eta \cdot \varphi(i, k) \cdot (x - w_i)$  ( $w_i$  is shifted towards  $x$ )
6. If convergence criterion met, STOP.  
Otherwise, narrow neighborhood function  $\varphi$  and learning parameter  $\eta$  and go to (2).

# Neighbourhood function

A neighborhood function  $\phi(i, k)$  indicates how closely neurons  $i$  and  $k$  in the output layer are connected to each other.

Usually, a Gaussian function on the distance between the two neurons in the layer is used:

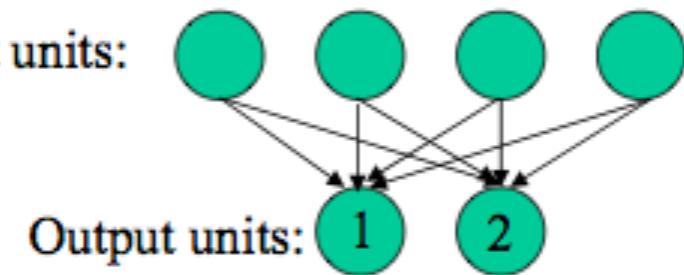


# Example SOM

- From Fausett (1994)
- $n = 4, m = 2$
- Training samples
  - i1: (1, 1, 0, 0)
  - i2: (0, 0, 0, 1)
  - i3: (1, 0, 0, 0)
  - i4: (0, 0, 1, 1)

*Network Architecture*

Input units:



Output units:

**What should we expect as outputs?**

# Distances between data samples

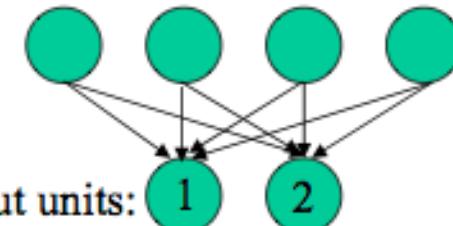
- Use Minkowski distance with  $r = 1$

- Training samples

- i1: (1, 1, 0, 0)
- i2: (0, 0, 0, 1)
- i3: (1, 0, 0, 0)
- i4: (0, 0, 1, 1)

*Network Architecture*

Input units:



Output units:

|    | i1 | i2 | i3 | i4 |
|----|----|----|----|----|
| i1 | 0  |    |    |    |
| i2 | 3  | 0  |    |    |
| i3 | 1  | 2  | 0  |    |
| i4 | 4  | 1  | 3  | 0  |

***What might we expect from the SOM?***

- Let Neighbourhood,  $\varphi(i, k) = 0$
- Learning rate  $\eta$   
 $\eta(t) = 0.6; 1 \leq t \leq 4$   
 $\eta(t) = 0.5 \eta(1); 5 \leq t \leq 8$   
 $\eta(t) = 0.5 \eta(5); 9 \leq t \leq 12$
- Initial weight matrix
  - Neuron 1, N1 [ .2, .6, .5, .9]
  - Neuron 2, N2 [ .8, .4, .7, .3]
- Weight update:  $w_i(t+1) := w_i(t) + \eta(t) \cdot (x - w_i)$
- **Problem:** Calculate the weights updates for the first four steps

- Training sample  $i_1$
- Distances (Euclidean) input pattern to all neurons
  - $D(1) = \sqrt{(.2-1)^2 + (.6-1)^2 + (.5-0)^2 + (.9-0)^2} = 1.86$
  - $D(2) = \sqrt{(.8-1)^2 + (.4-1)^2 + (.7-0)^2 + (.3-0)^2} = .98$
- Weights on winning neuron are updated
  - $w_i(t+1) := w_i(t) + .6 \cdot (x - w_i) = .4w_i(t) + .6x$
  - N1 [ .2, .6, .5, .9]
  - N2 [ .92, .76, .28, .12]

# Second weight update

- Training sample i2
- Distances input pattern to all neurons
  - $D(1) = (.2-0)^2 + (.6-0)^2 + (.5-0)^2 + (.9-1)^2 = .66$
  - $D(2) = (.92-0)^2 + (.76-0)^2 + (.28-0)^2 + (.12-1)^2 = 2.28$
  - N1 wins
- Weights on winning neuron are updated
  - $w_i(t+1) := w_i(t) + .6 \cdot (x - w_i) = .4w_i(t) + .6x$
  - N1 [ .08, .24, .20, .96]
  - N2 [ .92, .76, .28, .12]

**TODO:** Complete the example with training i3 and i4

- Once finished the first iteration (*also known as epoch*) the learning rate is updated, it will be .3
- Next, second iteration will start
- The process finishes when the number of iterations is reached
- After many iterations through the data set
  - N1 [ 0, 0, .5, 1]
  - N2 [ 1, .5, 0, 0]

**Did we get the clustering we expected?**

- Training samples

- i1: (1, 1, 0, 0)
- i2: (0, 0, 0, 1)
- i3: (1, 0, 0, 0)
- i4: (0, 0, 1, 1)

Sample: i1

- Weights

- N1 [ 0, 0, .5, 1]
- N2 [ 1, .5, 0, 0]

- Distance from unit1 weights

- $$(1-0)^2 + (1-0)^2 + (0-.5)^2 + (0-1.0)^2 = 1+1+.25+1=3.25$$

- Distance from unit2 weights

- $$(1-1)^2 + (1-.5)^2 + (0-0)^2 + (0-0)^2 = 0+.25+0+0=.25 \text{ (winner)}$$

Sample: i2

- Distance from unit1 weights

- $$(0-0)^2 + (0-0)^2 + (0-.5)^2 + (1-1.0)^2 = 0+0+.25+0 \text{ (winner)}$$

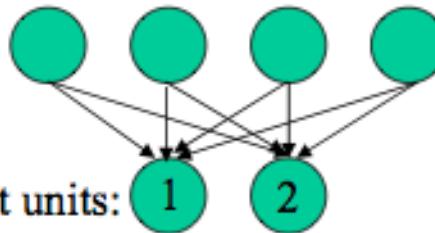
- Distance from unit2 weights

- $$(0-1)^2 + (0-.5)^2 + (0-0)^2 + (1-0)^2 = 1+.25+0+1=2.25$$

## *Network Architecture*

Input units:

Output units:



- Training samples

- i1: (1, 1, 0, 0)
- i2: (0, 0, 0, 1)
- i3: (1, 0, 0, 0)
- i4: (0, 0, 1, 1)

Sample: i3

- Weights

- N1 [ 0, 0, .5, 1]
- N2 [ 1, .5, 0, 0]

- Distance from unit1 weights

- $$(1-0)^2 + (0-0)^2 + (0-.5)^2 + (0-1.0)^2 = 1+0+.25+1=2.25$$

- Distance from unit2 weights

- $$(1-1)^2 + (0-.5)^2 + (0-0)^2 + (0-0)^2 = 0+.25+0+0=.25 \text{ (winner)}$$

Sample: i4

- Distance from unit1 weights

- $$(0-0)^2 + (0-0)^2 + (1-.5)^2 + (1-1.0)^2 = 0+0+.25+0 \text{ (winner)}$$

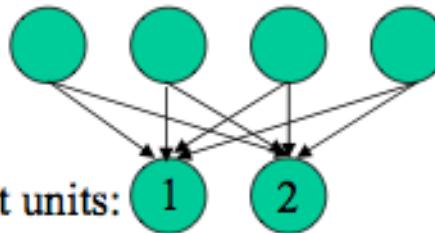
- Distance from unit2 weights

- $$(0-1)^2 + (0-.5)^2 + (1-0)^2 + (1-0)^2 = 1+.25+1+1=3.25$$

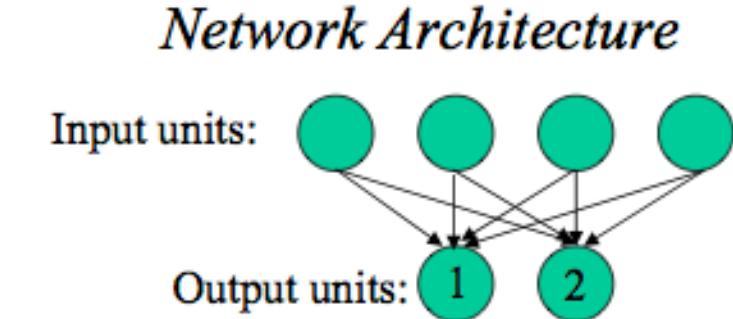
## *Network Architecture*

Input units:

Output units:



- Training samples
  - i1: (1, 1, 0, 0)
  - i2: (0, 0, 0, 1)
  - i3: (1, 0, 0, 0)
  - i4: (0, 0, 1, 1)
- Weights
  - N1 [ 0, 0, .5, 1]
  - N2 [ 1, .5, 0, 0]



- Samples i1, i3 clusters with neuron 2
- Samples i2, i4 clusters with neuron 1

- **Also known as Kohonen Maps**
  - It is an unsupervised ANN
  - Cooperative learning (through neighbourhood function)
  - Competitive learning (*winner takes most –if not all-*)
  - Discretization (in the form of network grids) and projection are simultaneously performed
  - SOM provides a topology preserving mapping from the high dimensional space to map units
  - A SOM has the capability to generalize
  - ***k-means*** is a special case of **SOM**

Arriving at this point:



## TO READ

An introduction to Self-Organizing Maps



- Read from page 299 to 315

In general all the chapter is interesting.

- You do not need to memorize the formulas, just to understand the concepts and what SOM is doing in its algorithm (the same contents that I have introduced you in the slides)



### 3. Multi-Dimensional Scaling (MDS)



# Introduction

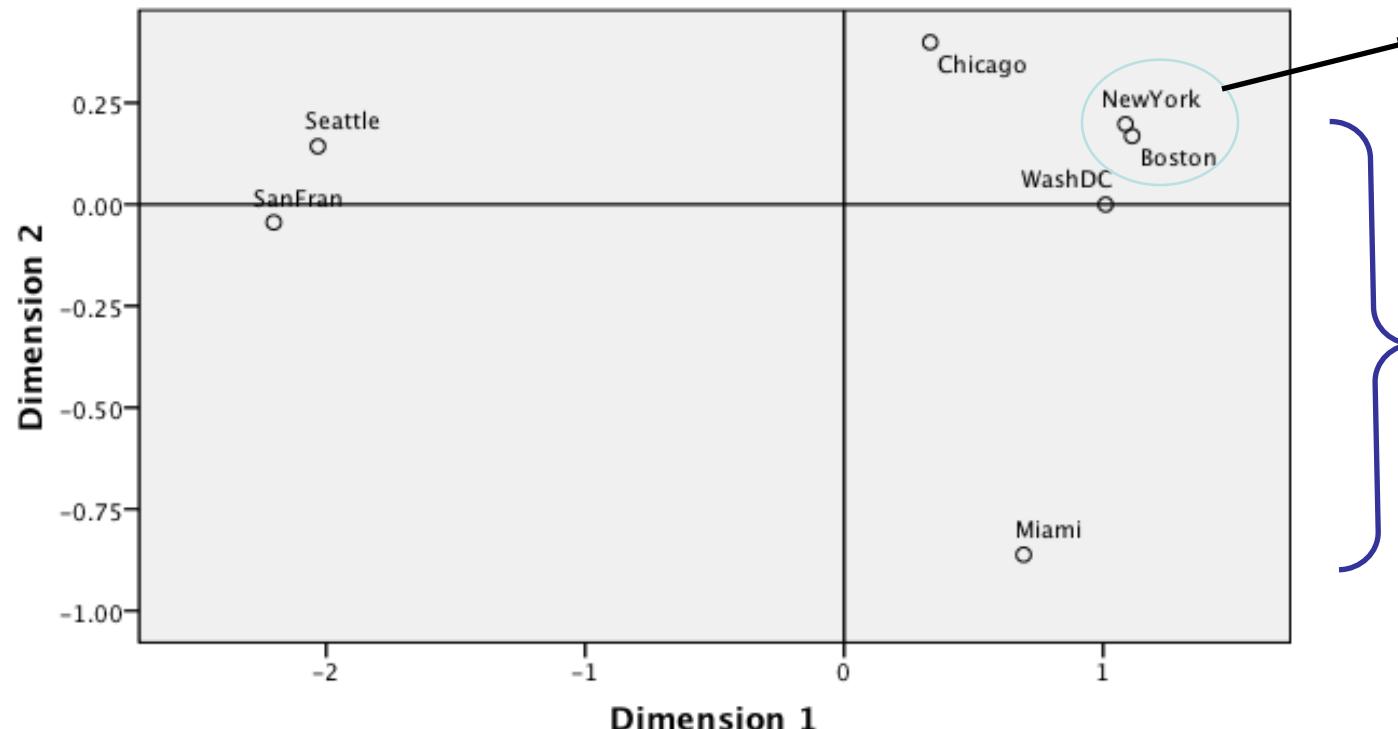
- “If you are interested in how certain objects relate to each other ... and if you would like to present these relationships in the form of a map then MDS is the technique you need”
- **The goal** of an MDS analysis is to find a spatial configuration of objects when all that is known is some measure of their general (dis)similarity.

- Generally regarded as **exploratory data analysis**
- **Reduces large amounts of data** into easy-to-visualize structures
- Attempts to **find structure** (visual representation) in a set of distance measures, e.g. dis/similarities, between objects/instances
  - Shows how variables/objects are related perceptually
- **How?** By assigning instances to specific locations in space
- Distances between points in space match dis/similarities as closely as possible:
  - Similar objects: Close points
  - Dissimilar objects: Far apart points

# MDS Example: City Distances

**Distances  
Matrix:  
Symmetric**

|   | 1       | 2    | 3    | 4    | 5    | 6    | 7    | 8    | 9    |      |
|---|---------|------|------|------|------|------|------|------|------|------|
|   | BOST    | NY   | DC   | MIAM | CHIC | SEAT | SF   | LA   | DENV |      |
| 1 | BOSTON  | 0    | 206  | 429  | 1504 | 963  | 2976 | 3095 | 2979 | 1949 |
| 2 | NY      | 206  | 0    | 233  | 1308 | 802  | 2815 | 2934 | 2786 | 1771 |
| 3 | DC      | 429  | 233  | 0    | 1075 | 671  | 2684 | 2799 | 2631 | 1616 |
| 4 | MIAMI   | 1504 | 1308 | 1075 | 0    | 1329 | 3273 | 3053 | 2687 | 2037 |
| 5 | CHICAGO | 963  | 802  | 671  | 1329 | 0    | 2013 | 2142 | 2054 | 996  |
| 6 | SEATTLE | 2976 | 2815 | 2684 | 3273 | 2013 | 0    | 808  | 1131 | 1307 |
| 7 | SF      | 3095 | 2934 | 2799 | 3053 | 2142 | 808  | 0    | 379  | 1235 |
| 8 | LA      | 2979 | 2786 | 2631 | 2687 | 2054 | 1131 | 379  | 0    | 1059 |
| 9 | DENVER  | 1949 | 1771 | 1616 | 2037 | 996  | 1307 | 1235 | 1059 | 0    |



**Cluster**

**Spatial Map**

**Dimensions**  
1: North/South  
2: East/West

# The Process of MDS: The Data

- Data of MDS: **similarities, dissimilarities, distances, or proximities** reflects amount of dis/similarity or distance between pairs of objects.
- Distinction between similarity and dissimilarity data dependent on type of scale used:
  - **Dissimilarity** scale: Low #=**high similarity** & High #=**high dissimilarity**.
  - **Similarity** scale: Opposite of dissimilarity.  
E.g. On a scale of 1-9 (1 being the same and 9 completely different) how similar are chocolate bars A and B? Dissimilarity scale.

MDS model classified according to:

1) Type of proximities:

- **Metric/quantitative:** Quantitative information/interval data about objects' proximities e.g. city distance
- **Non-metric/qualitative:** Qualitative information/nominal data about proximities e.g. rank order

2) Number of proximity matrices (distance, dis/similarity matrix).

- Proximity matrix is the input for MDS
- The above criteria yield:
  - 1) **Classical MDS:** One proximity matrix (metric or non-metric)
  - 2) **Replicated MDS:** Several matrices
  - 3) **Weighted MDS/Individual Difference Scaling:** Aggregate proximities and individual differences in a common MDS space

This course we will concentrate on the classical MDS.  
The next slides and the example come from this reading



## TO READ

An introduction to MDS

- From page 4 to page 12 (Introduction, Section 1, Introduction Section 2, and Section 2.1)

The remaining of the article is OPTIONAL.

# The MDS Model

- **Classical MDS** uses Euclidean principles to model data proximities in geometrical space, where distance ( $d_{ij}$ ) between points  $i$  and  $j$  is defined as:

$$d_{ij} = \sqrt{\sum (x_{ia} - x_{ja})^2}$$

$x_i$  and  $x_j$  specify coordinates of points  $i$  and  $j$  on dimension  $a$ , respectively.

- The modeled Euclidean distances are related to the observed proximities,  $\delta_{ij}$ , by some transformation/function ( $f$ ).
- Most MDS models assume that the data have the form:  
$$\delta_{ij} = f(d_{ij}) = \sqrt{\sum (x_{ia} - x_{ja})^2}$$
- All MDS algorithms are a variation of the above

# Classical MDS algorithm

We assume known a coordinate matrix  $X$

1. Set up the matrix of squared proximities,  $P^{(2)} = [p^2]$
2. Apply the double centering:  $B = -\frac{1}{2}JP^{(2)}J$   
using the matrix  $J = I - n^{-1}\mathbf{1}\mathbf{1}'$   
where  $n$  is the number of objects.
3. Extract the  $m$  largest positive eigenvalues of  $B$  and the corresponding  $m$  eigenvectors
4. A  $m$ -dimensional spatial configuration of the  $n$  objects is derived from the coordinate matrix  $X = E_m \Lambda_m^{1/2}$   
where  $E_m$  is the matrix of  $m$  eigenvectors and  $\Lambda_m$  is the diagonal matrix of  $m$  eigenvalues

- Proximity matrix

- Distances

- between 4 cities

|     |     |     |     |     |
|-----|-----|-----|-----|-----|
|     | cph | aar | ode | aal |
| cph | 0   | 93  | 82  | 133 |
| aar | 93  | 0   | 52  | 60  |
| ode | 82  | 52  | 0   | 111 |
| aal | 133 | 60  | 111 | 0   |

- Step 1 – Matrix of squared proximities

$$\mathbf{P}^{(2)} = \begin{bmatrix} 0 & 8649 & 6724 & 17689 \\ 8649 & 0 & 2704 & 3600 \\ 6724 & 2704 & 0 & 12321 \\ 17689 & 3600 & 12321 & 0 \end{bmatrix}$$

- Step 2- Apply the double centering

$$\mathbf{J} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - 0.25 \times \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0.75 & -0.25 & -0.25 & -0.25 \\ -0.25 & 0.75 & -0.25 & -0.25 \\ -0.25 & -0.25 & 0.75 & -0.25 \\ -0.25 & -0.25 & -0.25 & 0.75 \end{bmatrix}.$$

Applying  $\mathbf{J}$  to  $\mathbf{P}^{(2)}$  yields the double centered matrix  $\mathbf{B}$

$$\mathbf{B} = -\frac{1}{2}\mathbf{JP}^{(2)}\mathbf{J} = \begin{bmatrix} 5035.0625 & -1553.0625 & 258.9375 & -3740.938 \\ -1553.0625 & 507.8125 & 5.3125 & 1039.938 \\ 258.9375 & 5.3125 & 2206.8125 & -2471.062 \\ -3740.9375 & 1039.9375 & -2471.0625 & 5172.062 \end{bmatrix}.$$

- Step 3- Extract the two largest eigenvalues and the corresponding eigenvectors

$$\lambda_1 = 9724.168, \lambda_2 = 3160.986$$

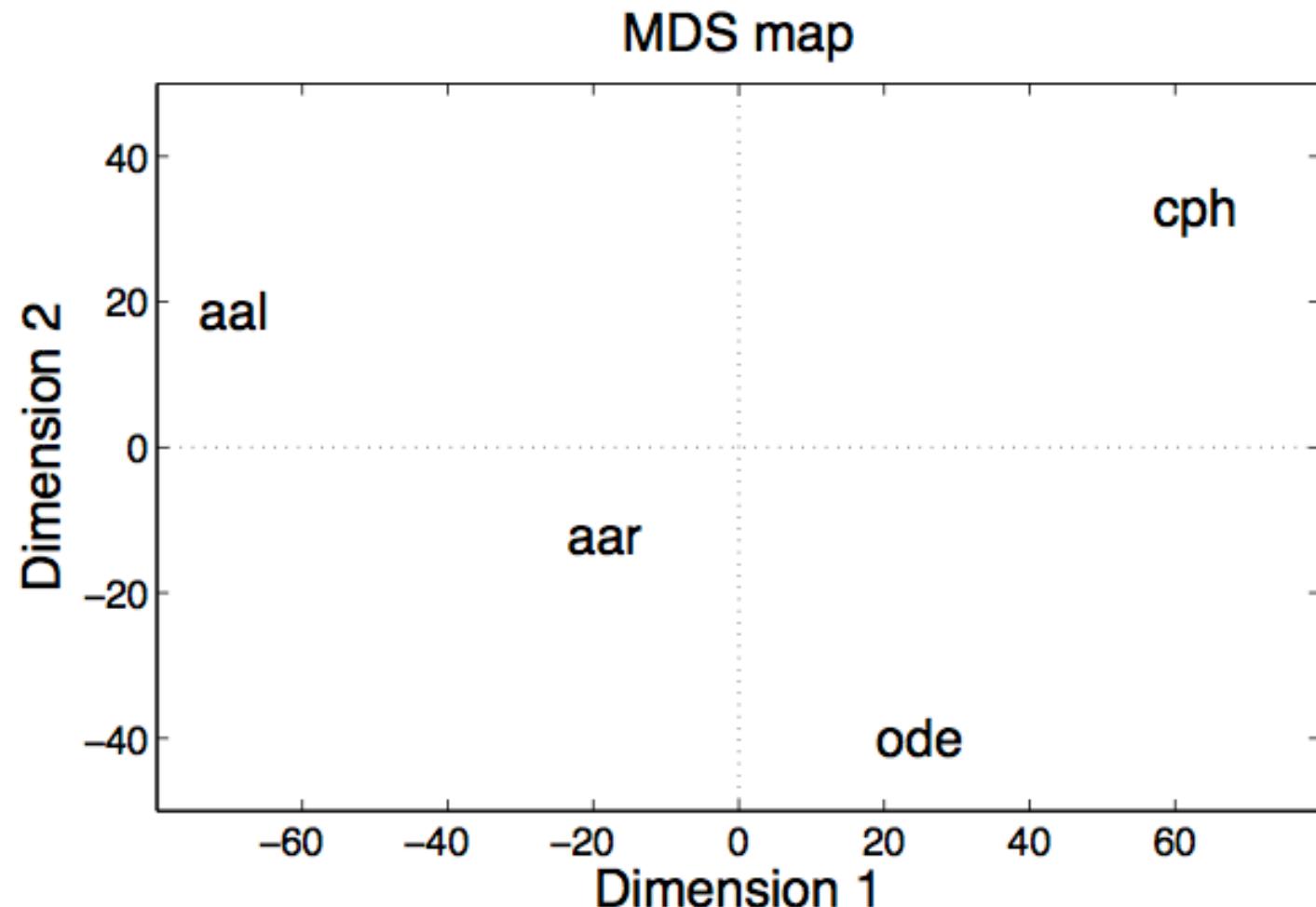
$$\mathbf{e}_1 = \begin{pmatrix} -0.637 \\ 0.187 \\ -0.253 \\ 0.704 \end{pmatrix}, \quad \mathbf{e}_2 = \begin{pmatrix} -0.586 \\ 0.214 \\ 0.706 \\ -0.334 \end{pmatrix}$$

- Step 4- the coordinates of the cities are obtained by multiplying eigenvalues and vectors

$$\mathbf{X} = \begin{bmatrix} -0.637 & -0.586 \\ 0.187 & 0.214 \\ -0.253 & 0.706 \\ 0.704 & -0.334 \end{bmatrix} \begin{bmatrix} \sqrt{9724.168} & 0 \\ 0 & \sqrt{3160.986} \end{bmatrix} = \begin{bmatrix} -62.831 & -32.97448 \\ 18.403 & 12.02697 \\ -24.960 & 39.71091 \\ 69.388 & -18.76340 \end{bmatrix}$$

# Classical MDS Example

- The result is ....



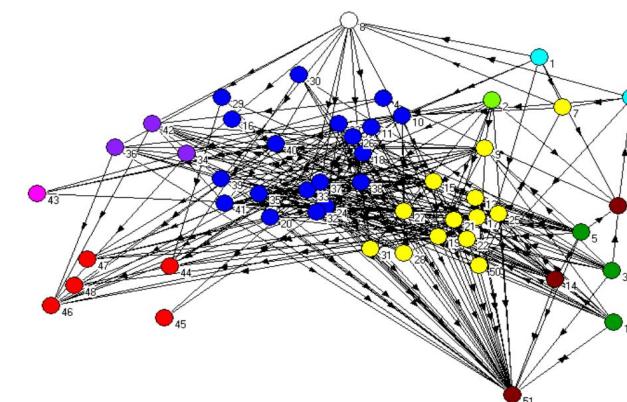
## MDS Map/Perceptual Map/Spatial Representation:

1) **Clusters:** Groupings in a MDS spatial representation.

- These may represent a domain/subdomain.

2) **Dimensions:** Hidden structures in data. Ordered groupings that explain similarity between items.

- Axes are meaningless and orientation is arbitrary.
- In theory, there is no limit to the number of dimensions.
- In reality, the number of dimensions that can be perceived and interpreted is limited.





# “Advantages” of MDS

- Does not require assumptions of linearity, metricity, or multivariate normality
- Can be used to model nonlinear relationships
- Dimensionality “solution” can be obtained from individuals; gives insight into how individuals differ from aggregate data
- Reveals dimensions without the need for defined attributes
- Dimensions that emerge from MDS can be incorporated into regression analysis to assess their relationship with other variables

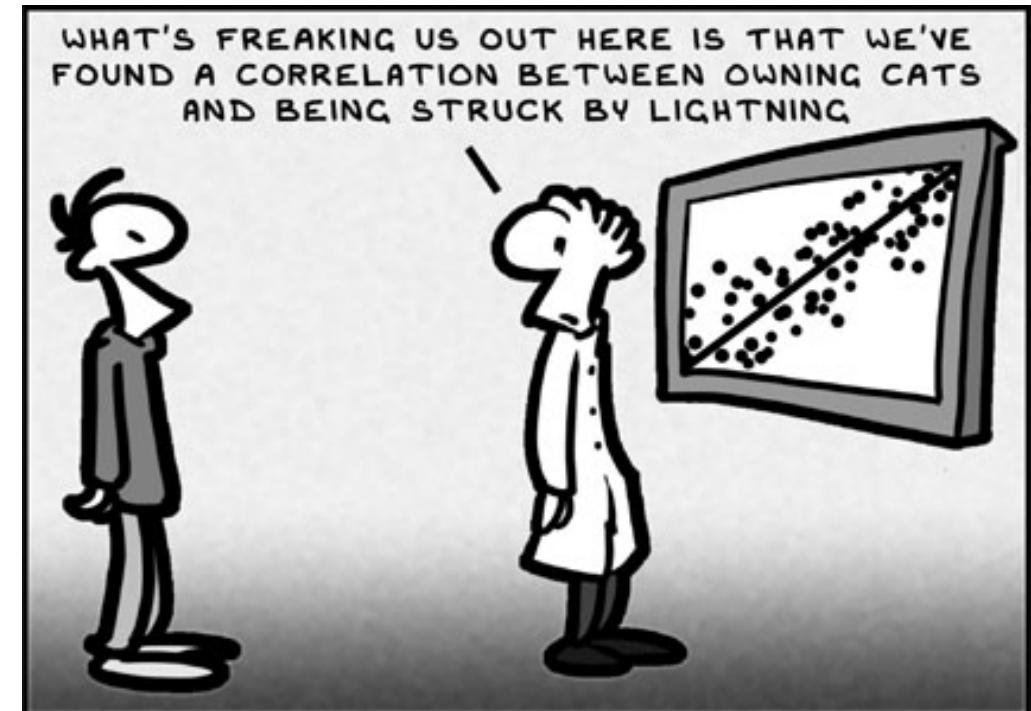
# “Disadvantages” of MDS

- Provides a global measure of dis/similarity but does not provide much insight into subtleties
- Increased dimensionality: Difficult to represent and decreases intuitive understanding of the data. As such, the model of the data becomes as complicated as the data itself.
- Determination of meanings of dimensions is subjective.

- **MDS:** technique used in data visualisation for exploring similarities or dissimilarities in data. An MDS algorithm starts with a matrix of item-item similarities, then assigns a location of each item in a low-dimensional space, suitable for graphing or 3D visualisation.
- **Taxonomy:**
  - Metric multidimensional scaling -- assumes the input matrix is just an item-item distance matrix. Analogous to PCA, an eigenvector problem is solved to find the locations that minimize distortions to the distance matrix. Its goal is to find a Euclidean distance approximating a given distance.
  - Generalized multidimensional scaling (GMDS) -- A superset of metric MDS that allows for the target distances to be non-Euclidean.
  - Non-metric multidimensional scaling -- It finds a non-parametric monotonic relationship between the dissimilarities in the item-item matrix and the Euclidean distance between items, and the location of each item in the low-dimensional space



*Questions?*





## Week 10

# Course. Introduction to Machine Learning **Theory 10. Visualization**

Dr. Maria Salamó Llorente  
[maria.salamo@ub.edu](mailto:maria.salamo@ub.edu)

Dept. Mathematics and Informatics,  
Faculty of Mathematics and Informatics,  
University of Barcelona (UB)