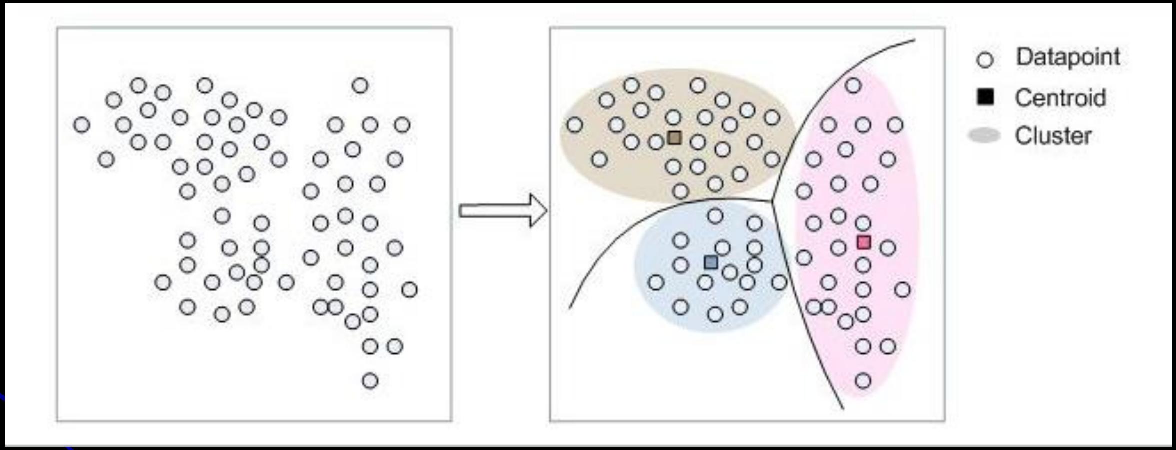
Machine learning in computer vision

Lesson 10

Nonhierarchical methods

the space is divided into one set of clusters



K-means clustering

K clusters

Minimizes total intra-cluster scatter (within sum of squares - WSS)

$$W = \sum_{k=1}^{K} \sum_{x_i \in C_k} \sum_{x_j \in C_k} \left\| x_i - x_j \right\|^2 = \sum_{k=1}^{K} 2N_k \sum_{x_i \in C_k} \left\| x_i - m_k \right\|^2 = \sum_{k=1}^{K} WSS_k$$

m_k centroid of cluster k

N_k number of points in cluster k

K-Means Algorithm

Initialization:

Randomly place K points into the space represented by the objects that are being clustered. These points represent initial group centroids.

K-Means Algorithm

Assign objects to the group that has the closest centroid

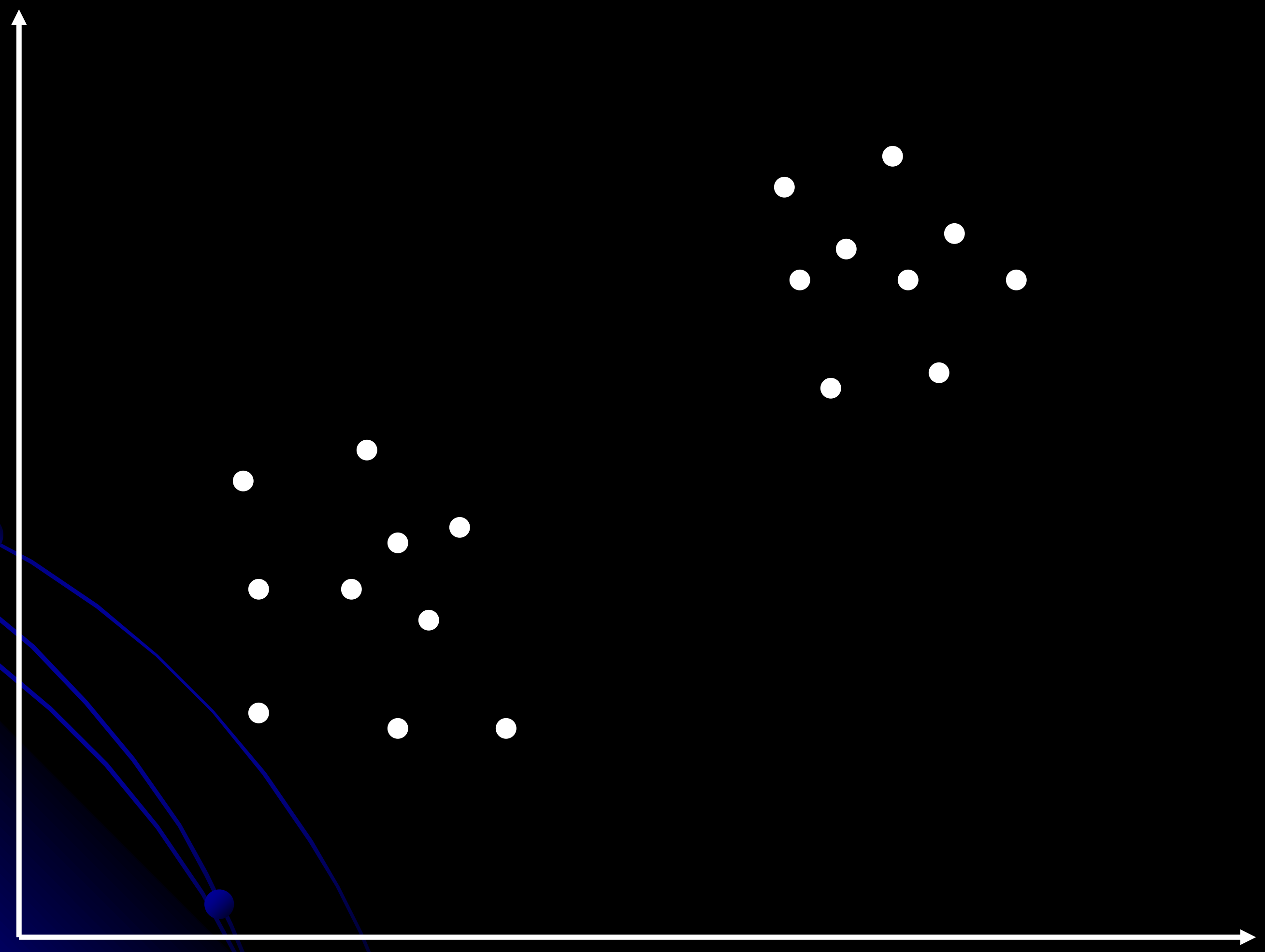
$$C(x) = \arg\min_{k} \|x - m_k\|^2$$

When all objects have been assigned, recalculate the positions of the K centroids

$$m_k = \frac{\sum_{x:C(x)=k} x}{N_k}, k = 1, \ldots K$$

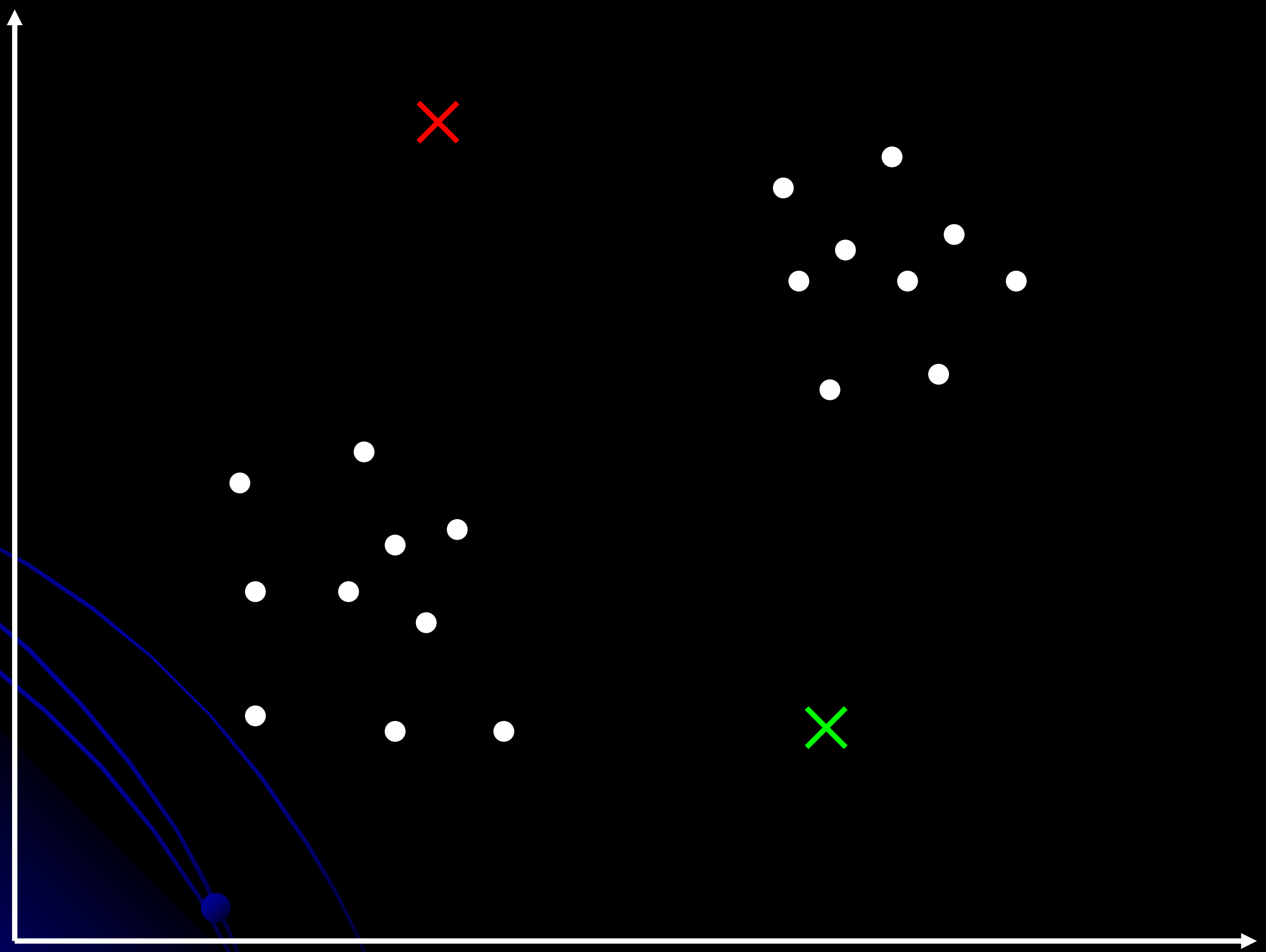
Repeat until the stopping criteria is met.
(MSE < threshold, or no change in clustering)

K-Means Clustering



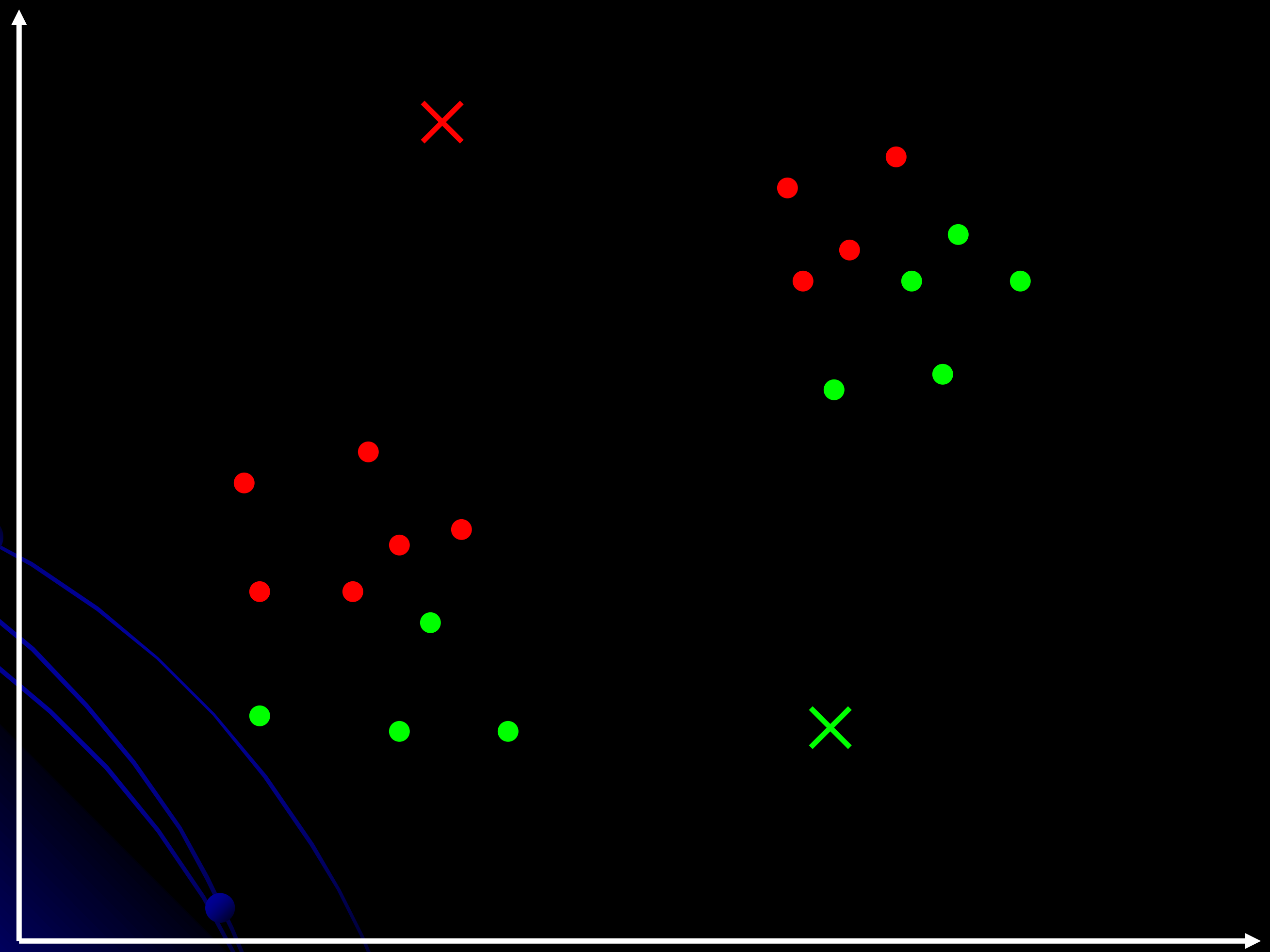
Initialization

K-Means Clustering



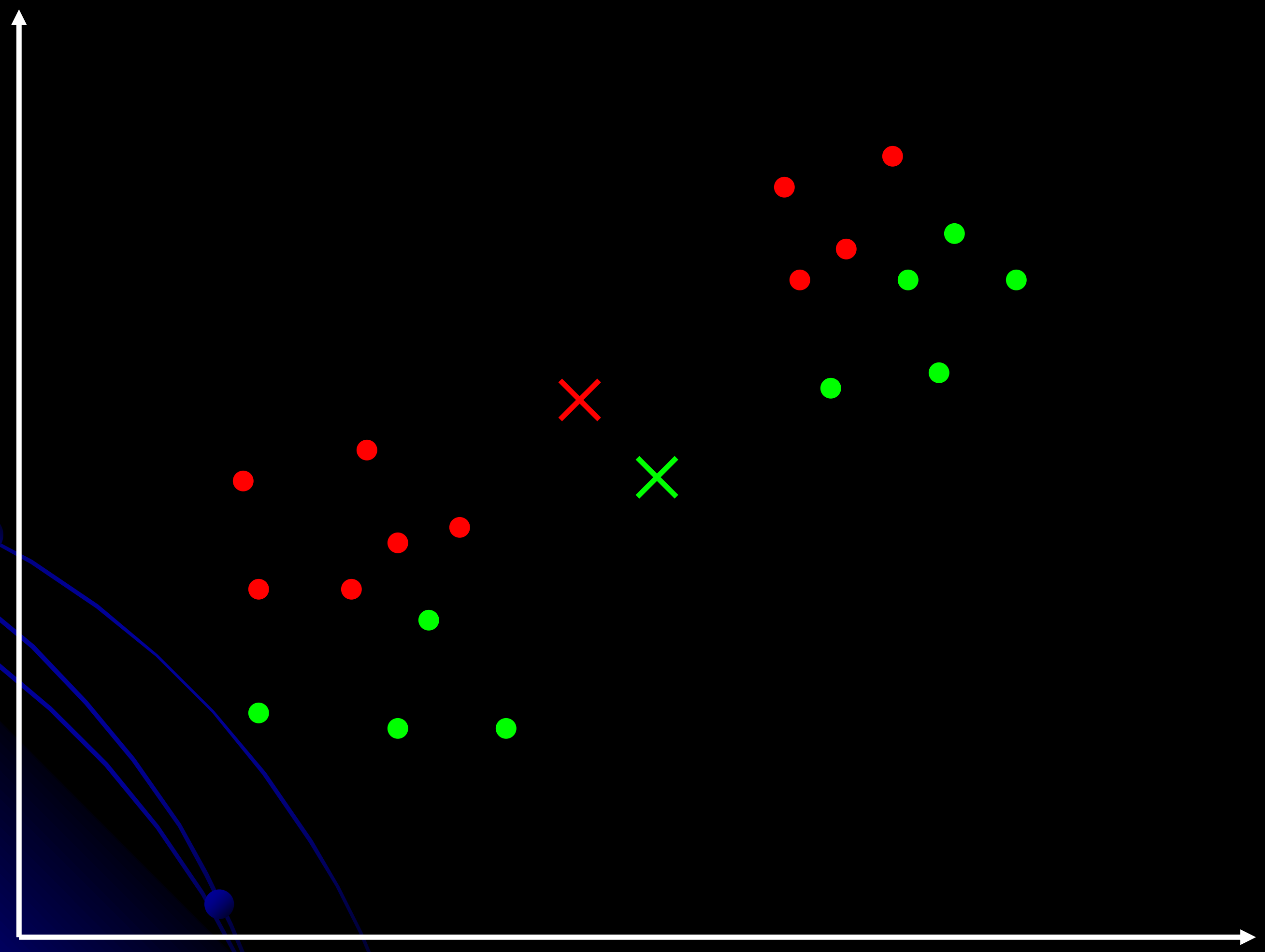
Initialization

K-Means Clustering



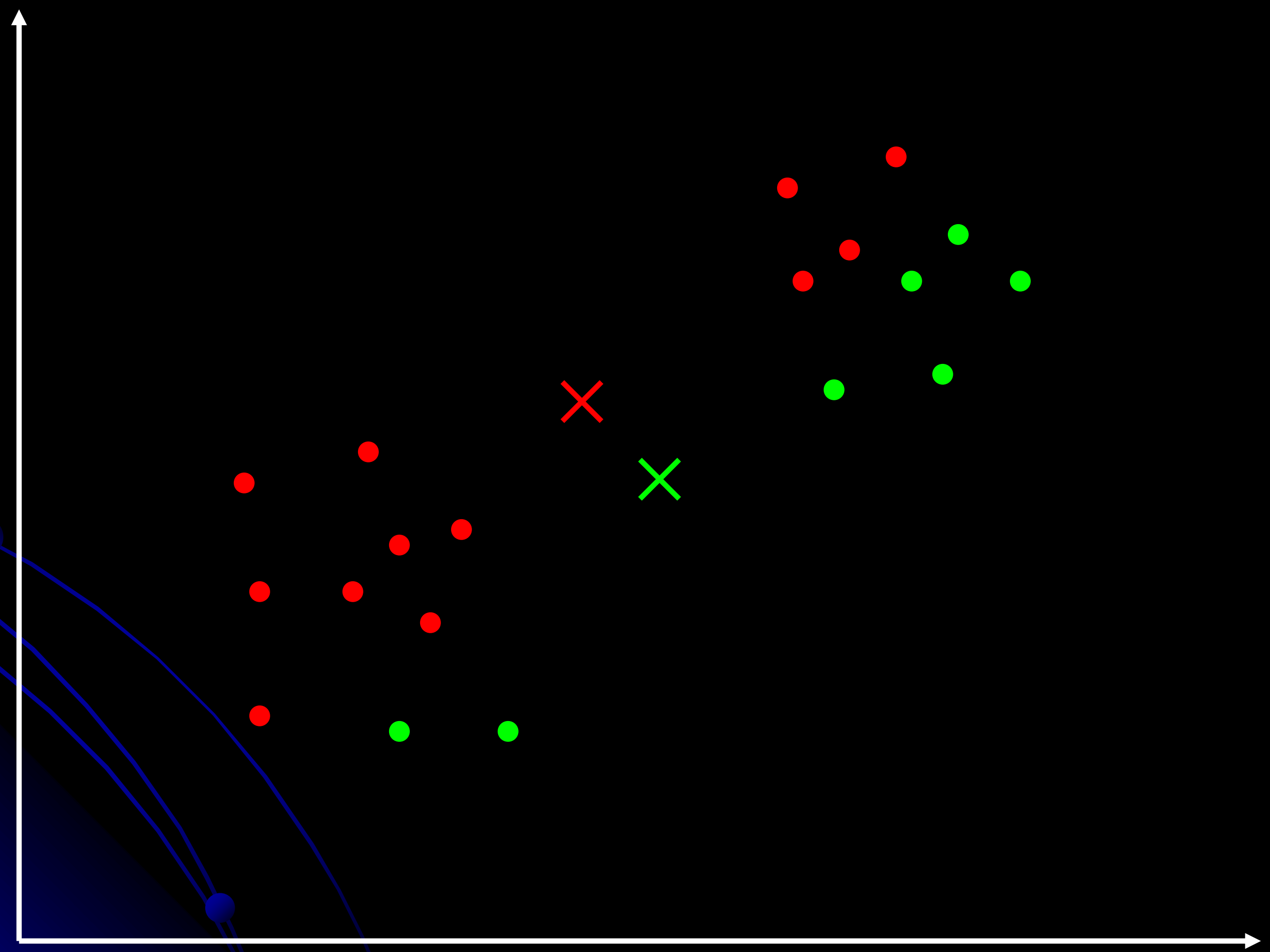
Assign objects

K-Means Clustering



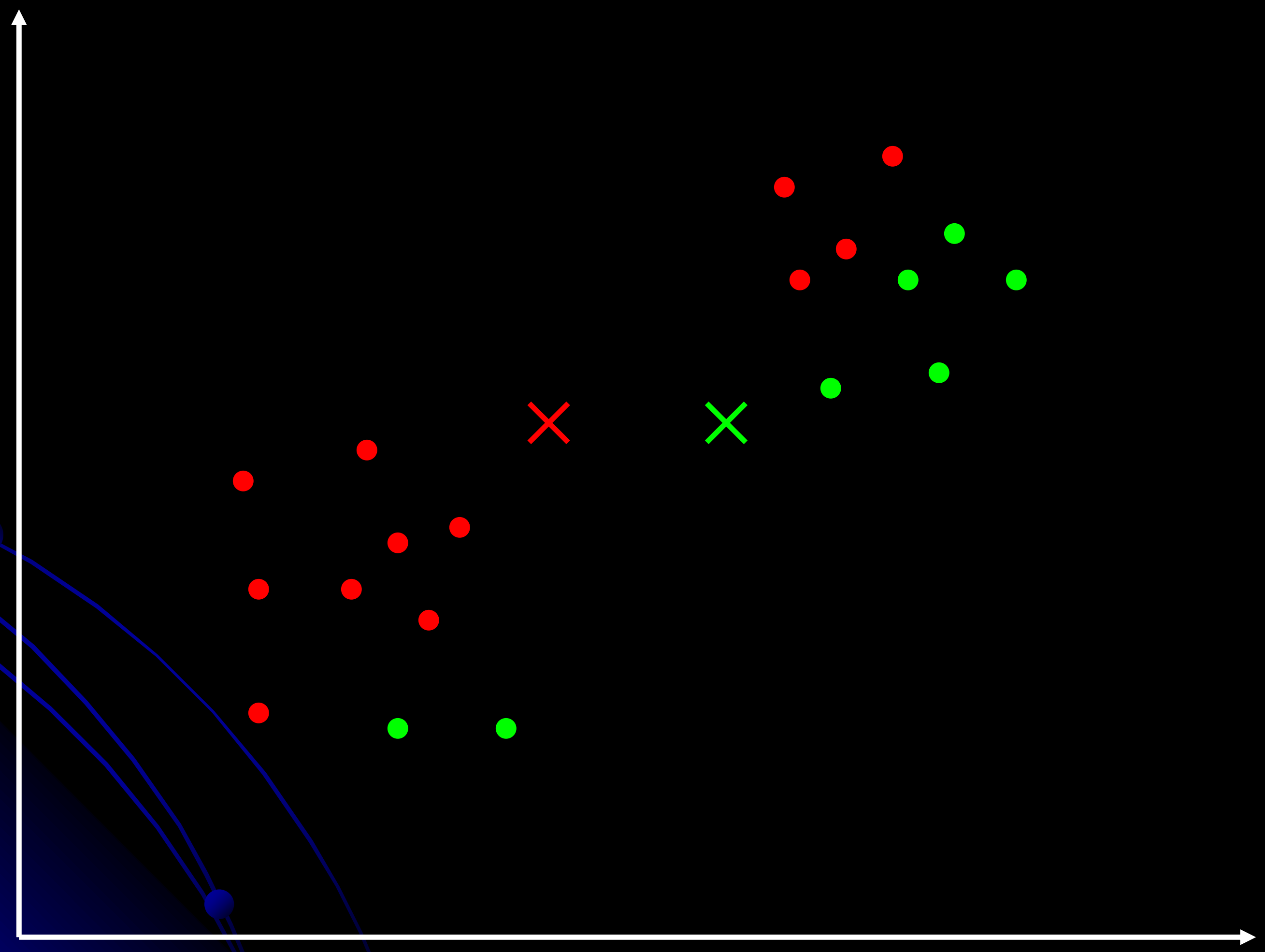
Recalculate centroids

K-Means Clustering



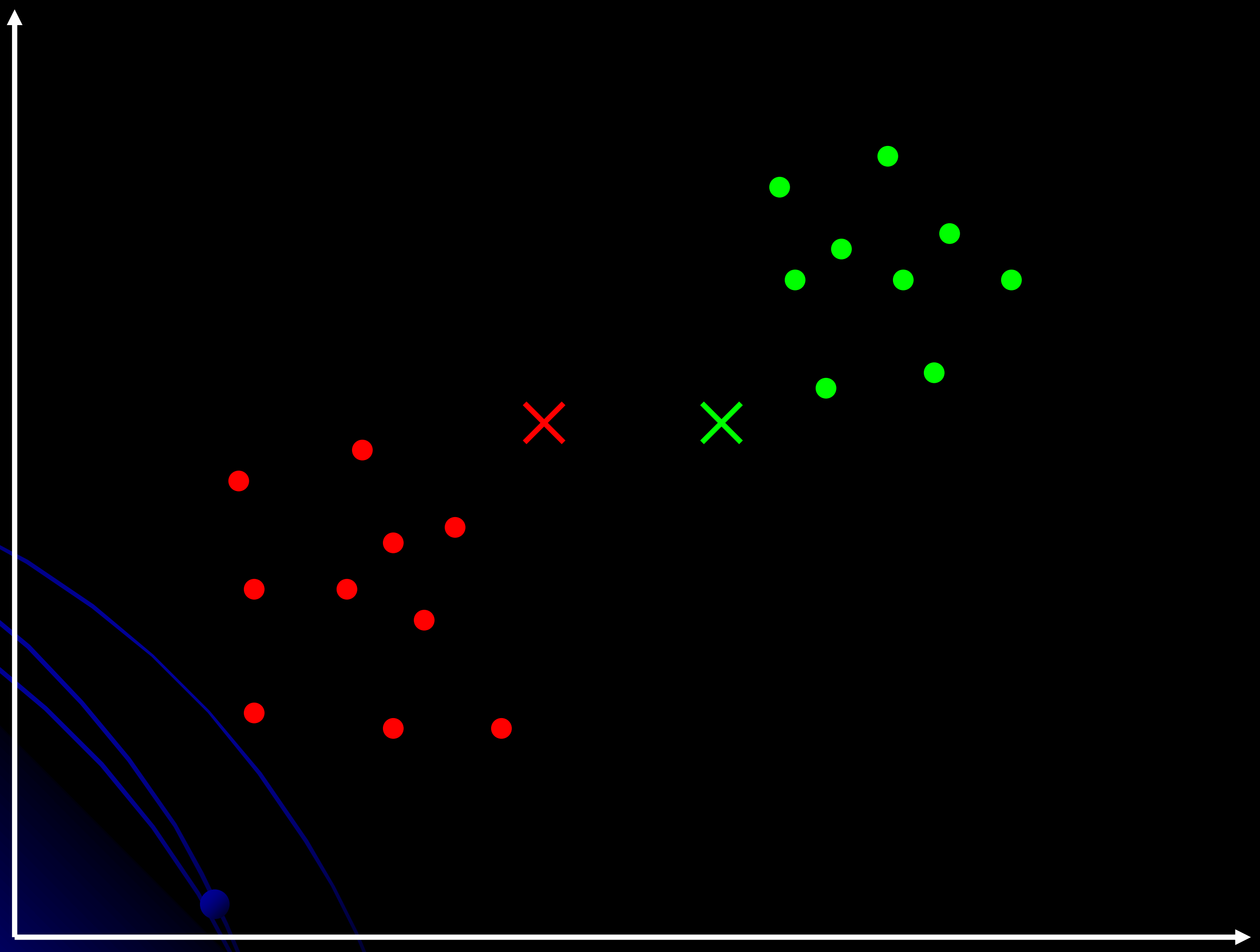
Assign objects

K-Means Clustering



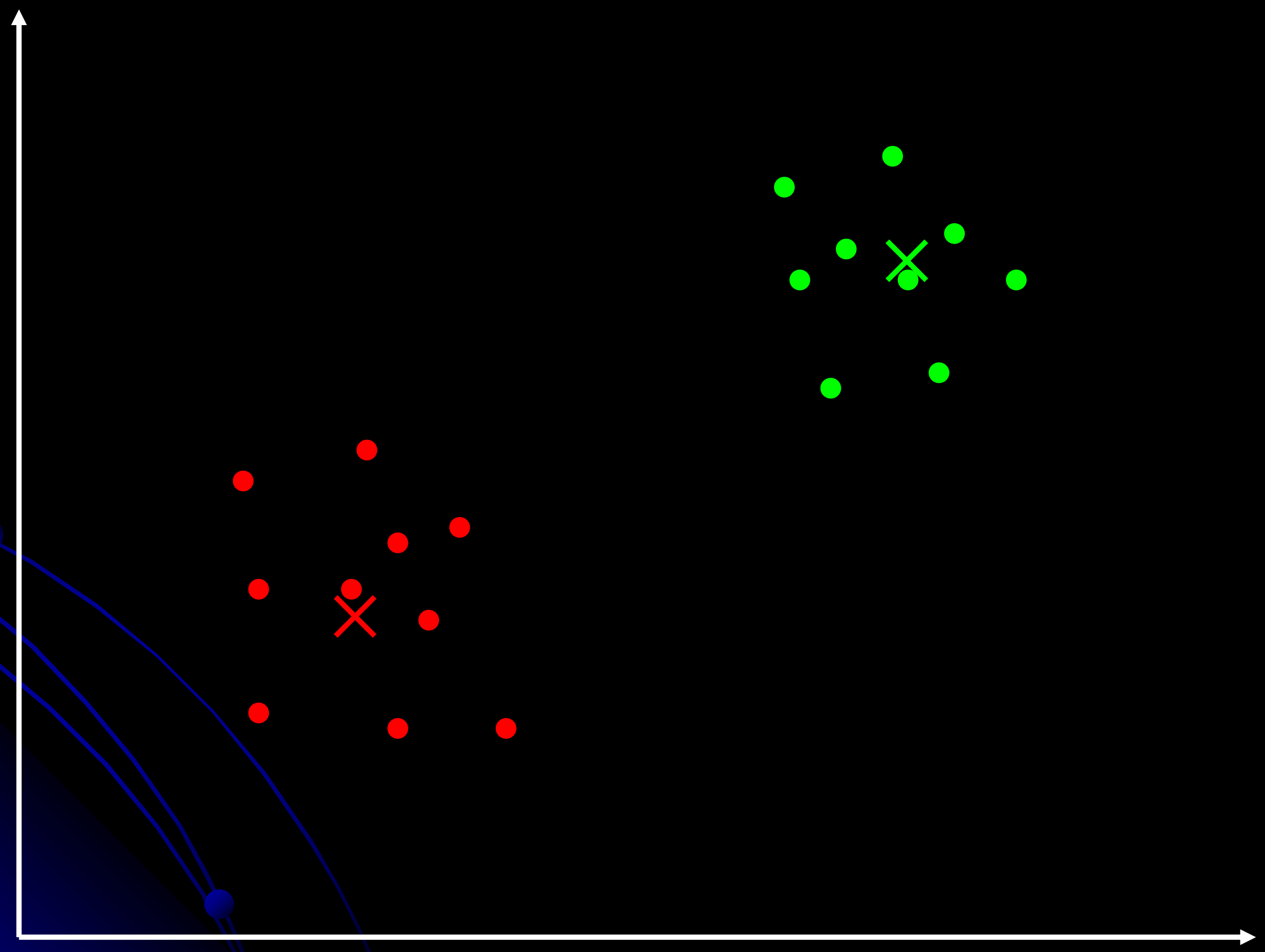
Recalculate centroids

K-Means Clustering



Assign objects

K-Means Clustering



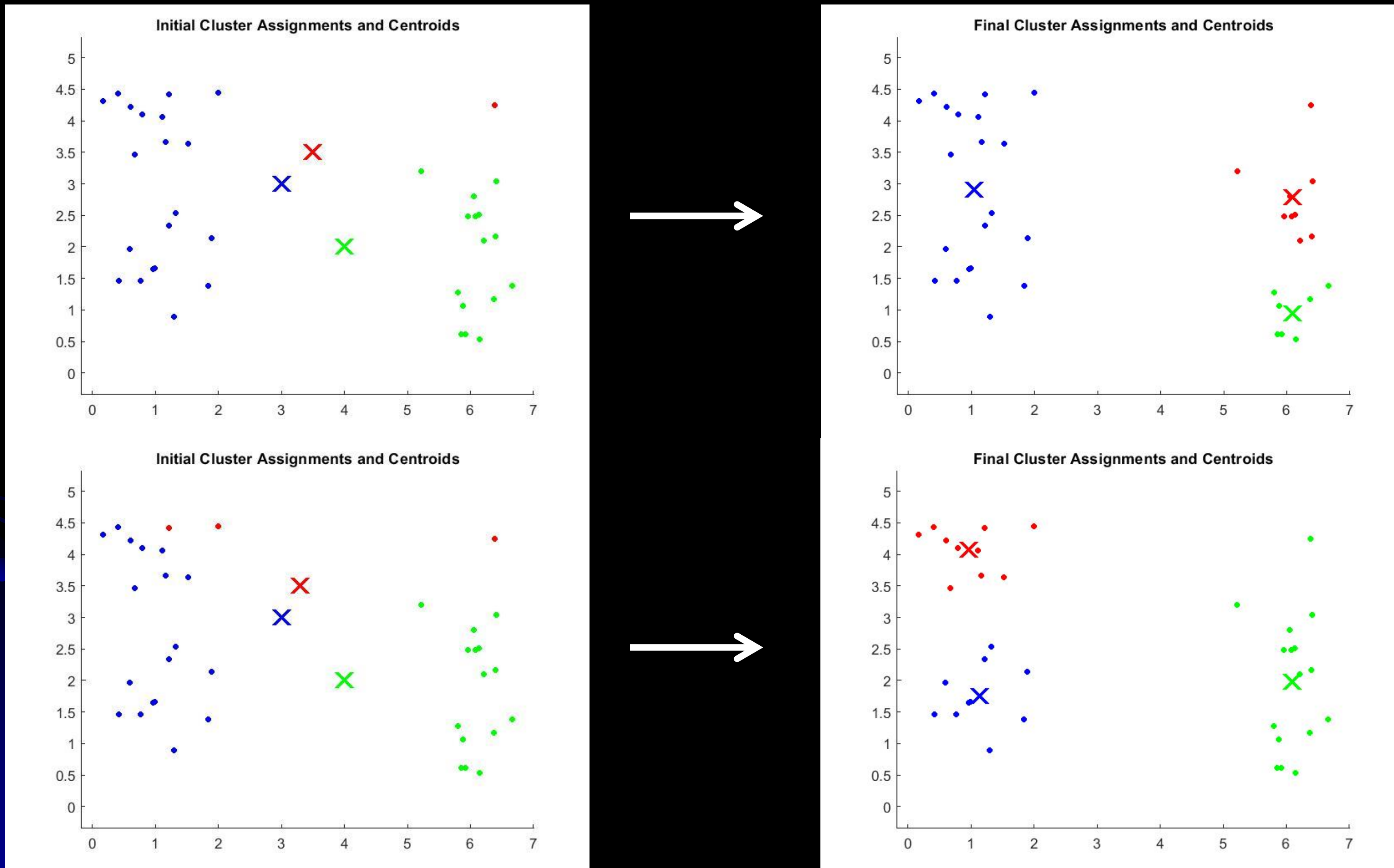
Recalculate centroids

END

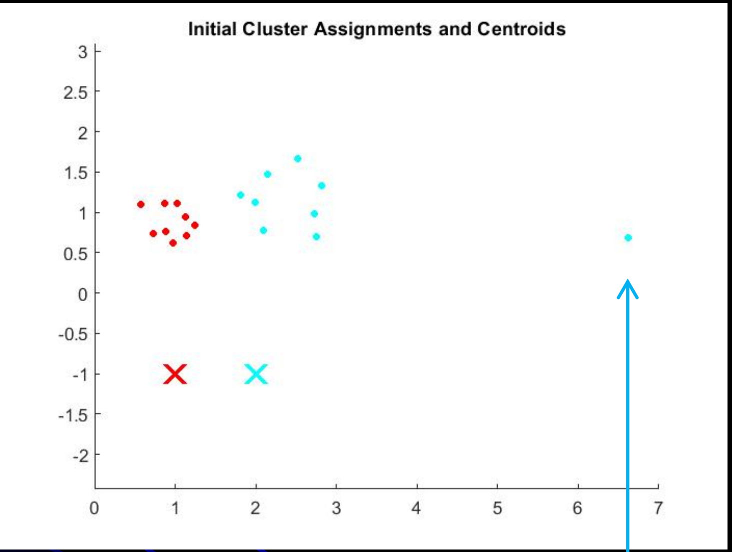
K-Means Clustering

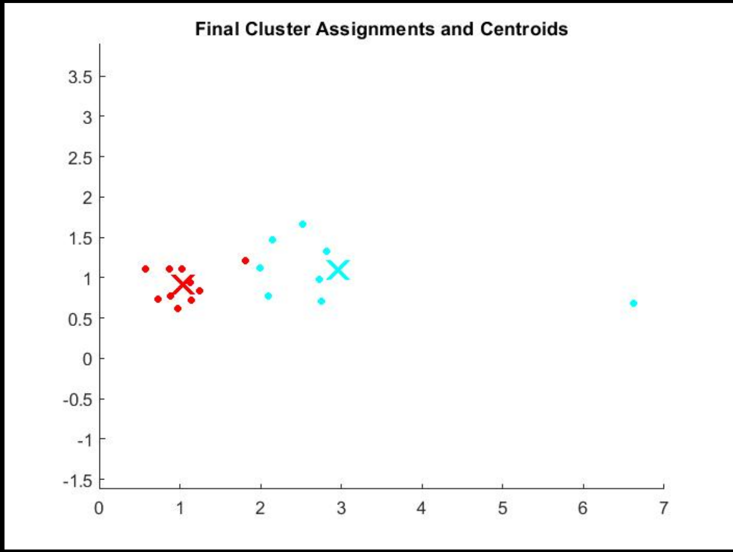
- Guaranteed to converge
- Guaranteed to achieve local optimum, not necessarily global optimum
- Sensitive to noise and outlier data points
- Clusters are sensitive to initial assignment of centroids (not a deterministic algorithm)
- Clusters can be inconsistent from one run to another

Initialization



Outliers



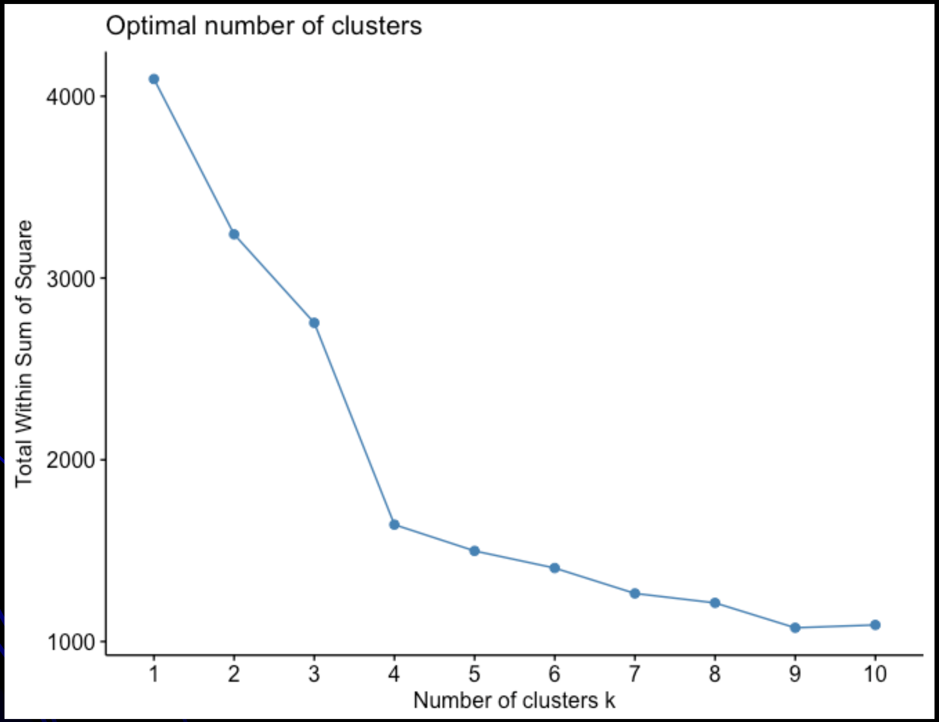


Outlier

Choice of K

Many methods

Elbow point: compute total WSS

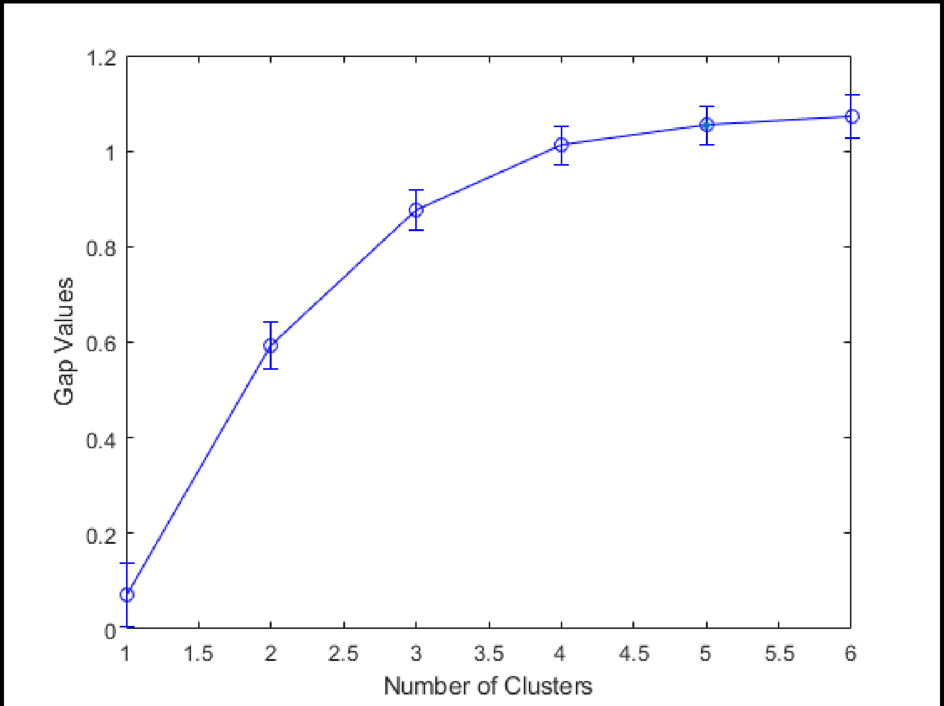


Choice of K

Gap value

$$\text{Gap}_N(K) = E_N^*\{W_K\} - \log W_K$$

$$W_K = \sum_{k=1}^{K} \frac{1}{2N_k} WSS_k$$



Choice of K

The silhouette value:

a measure of how similar a point is to points in its own cluster, when compared to points in other clusters

$$S(i) = \frac{b(i) - a(i)}{\max\big(a(i), b(i)\big)}$$

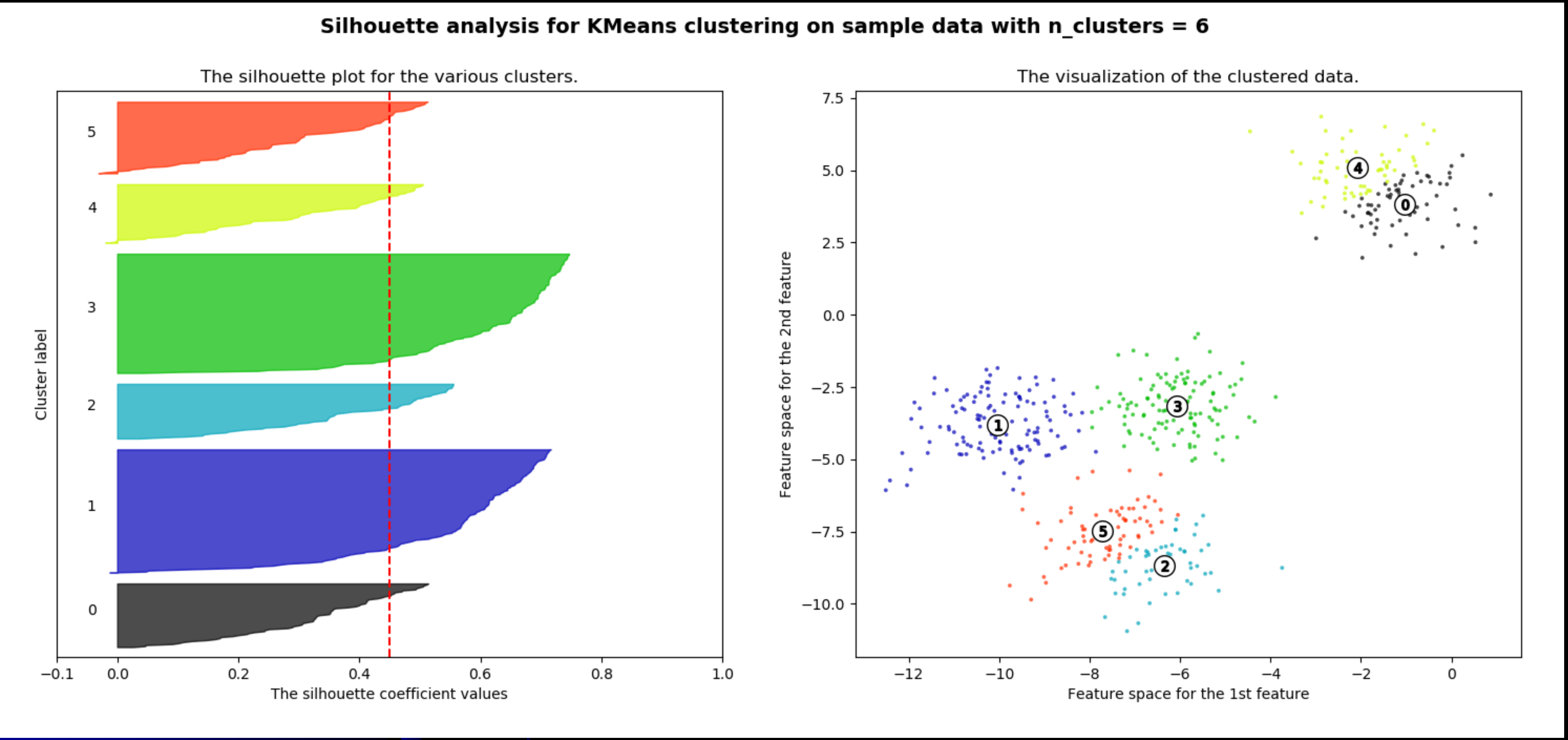
$a(i)$ – average distance from the i-th point to the other points in the same cluster as i

$b(i)$ – minimum average distance from the i-th point to points in a different cluster, minimized over clusters

Try different Ks, compute average silhouette

Choice of K

The silhouette plot



https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html

K-medoids

- instead of means, use medoids of each cluster
- Medoid – object already in the set (e.g. existing color)
- Mean – “artificial” object

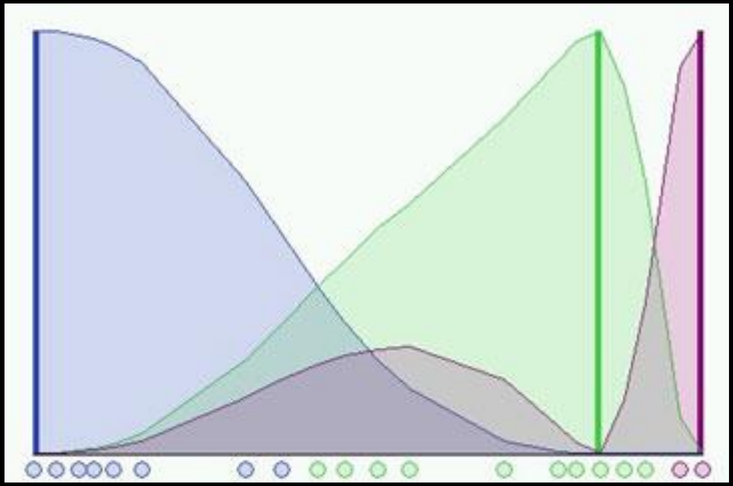
K-medians

- instead of means, use medians in each dimension
- object might not be in the set

Fuzzy C-means

Soft membership function

$$\sum_{k=1}^{K} \sum_{x_i \in C_k} w_{ik}^{f} \|x_i - m_k\|^2$$



https://matteucci.faculty.polimi.it/Clustering/tutorial_html/cmeans.html

Buckshot

hierarchical agglomerative clustering (HAC) and K-Means clustering

First randomly take a sample of instances of size $\sqrt{n}$

Run group-average HAC on this sample, which takes only O(n) time

Use the results of HAC as initial seeds for K-means

Overall algorithm is O(n) and avoids problems of bad seed selection

https://nlp.stanford.edu/IR-book/information-retrieval-book.html

Bisecting K-means

Divisive hierarchical clustering method using K-means

For I=1 to K-1 do {

    Pick a leaf cluster C to split

    For J=1 to ITER do {

        Use K-means to split C into two sub-clusters, C_1 and C_2

        }

    Choose the best of the above splits and make it permanent

    }

http://glaros.dtc.umn.edu/gkhome/node/157

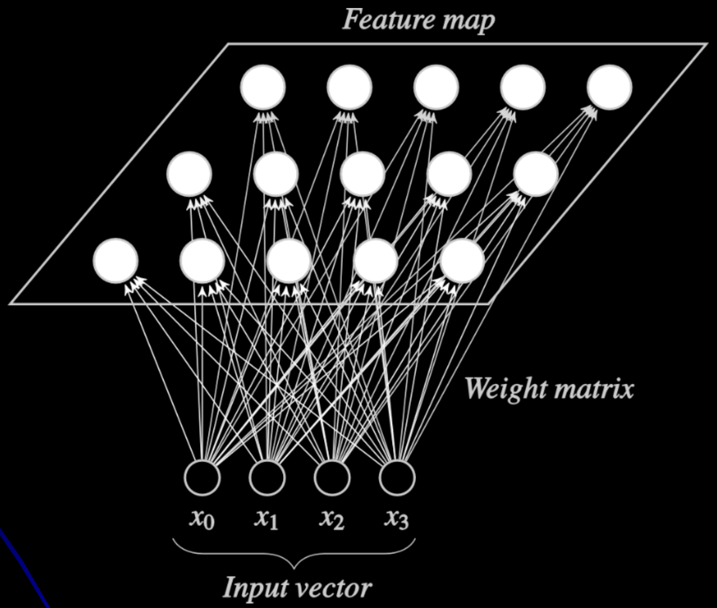
Self-Organizing Maps

SOM – Kohonen nets

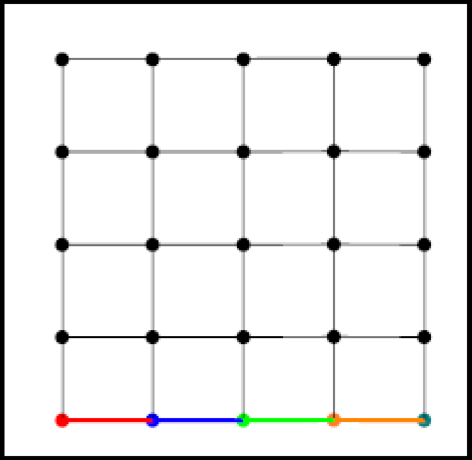
Self-Organizing Maps

Neurons form a lattice

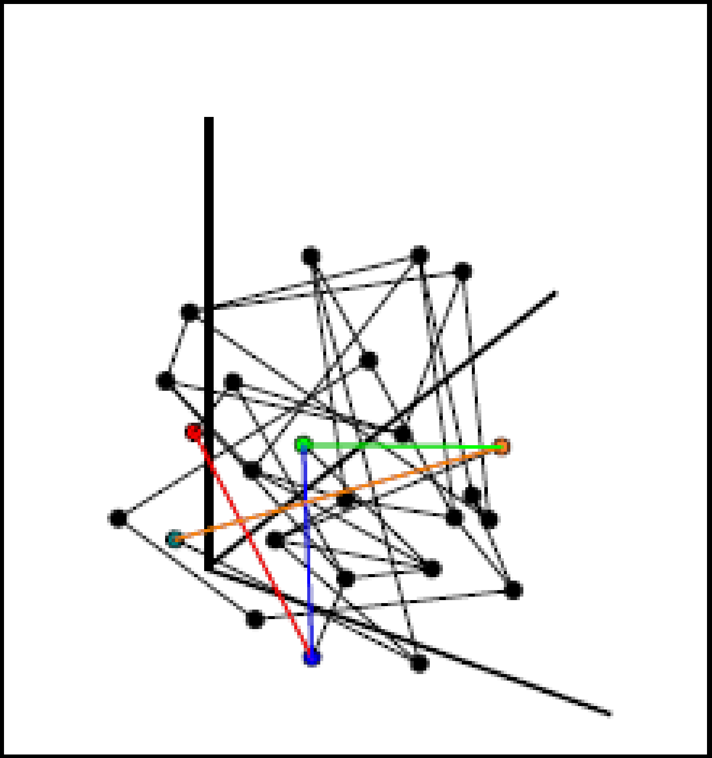
Input data connected with all neurons



Two spaces of SOM



SOM lattice
Topological structure



Weight space
Same dimensionality as input space

SOM learning

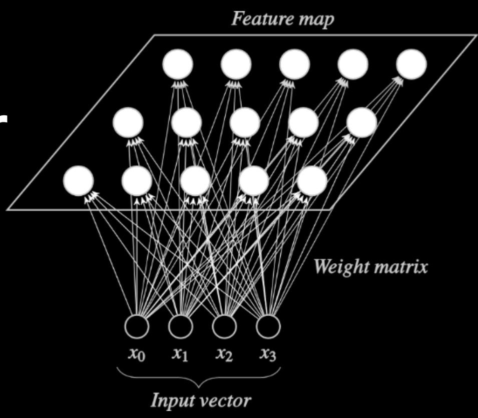
Winner-takes-all algorithm:

The closest node is updated

Algorithm:

1. Randomize the map's nodes weight

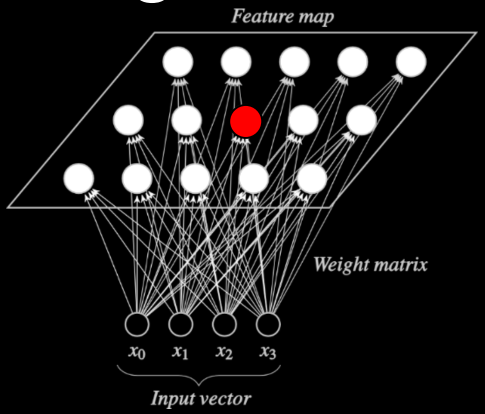
2. Select randomly one input vector



SOM learning

3. Find the closest node: best matching unit

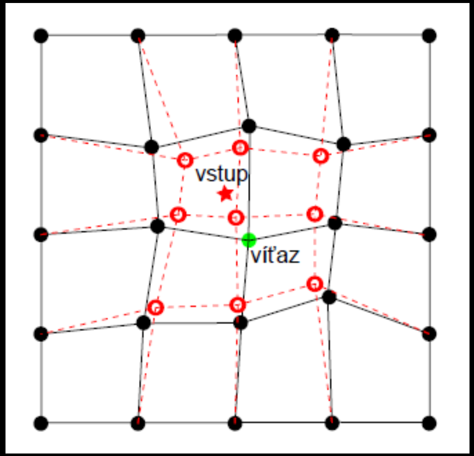
$$i^* = \arg\min_i \|\mathbf{x} - \mathbf{w}_i\|$$



Closest using Euclid distance

4. The weight of this node is updated

Winner-takes-all



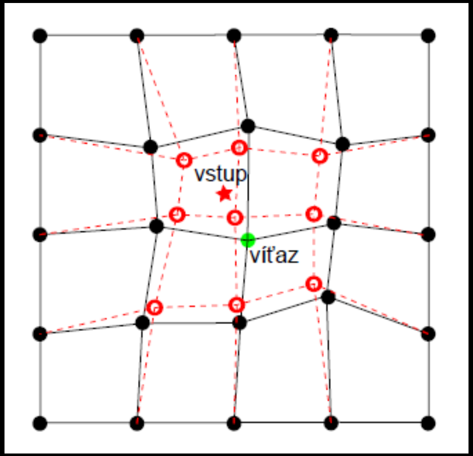
SOM learning

5. The weights of the adjacent nodes are also updated, by not to the same degree

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \eta(t)O(i,i^*,t)\|\mathbf{x} - \mathbf{w}_i(t)\|$$

$O(i,i^*,t)$ – neighborhood specification

$\eta(t)$ – learning rate

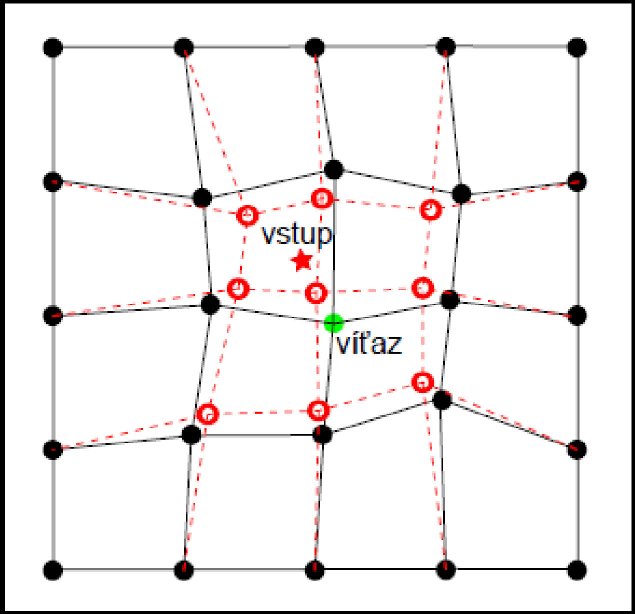


SOM learning

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \eta(t)O(i,i^*,t)\|\mathbf{x} - \mathbf{w}_i(t)\|$$

Cooperation phase

This is what ensures the similarity of weights between contiguous nodes



SOM learning

6. Reduce the intensity of the update progressively

Adaptation phase

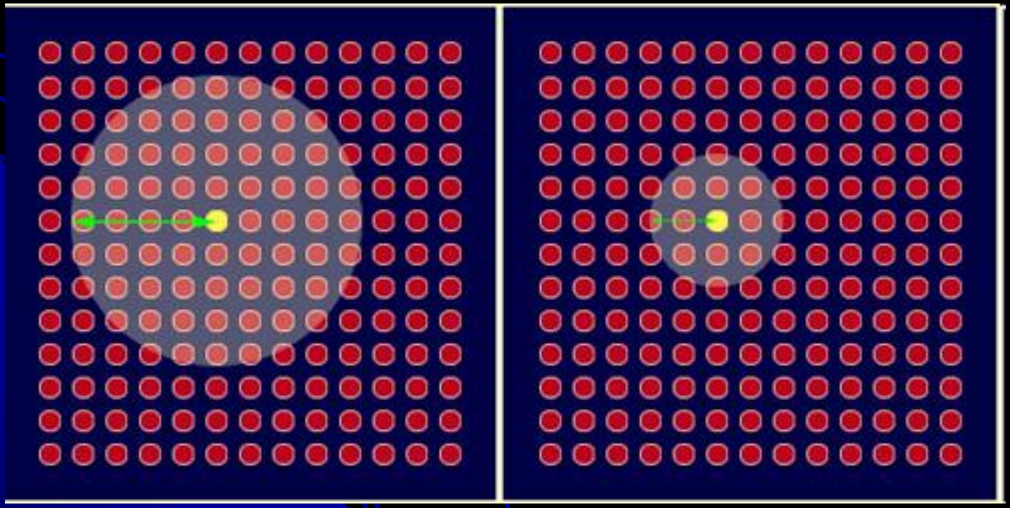
At first, high learning rate, move quickly to the solution; at the end, small learning rate, to avoid oscillations.

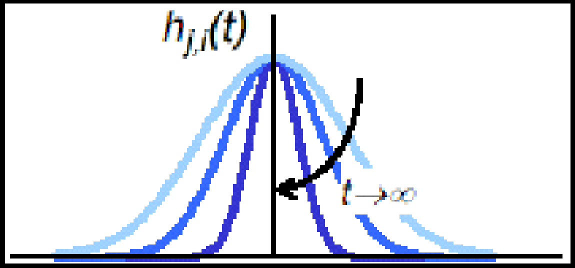
7. Repeat 1 to 6 for T_{max} iterations

Neighborhood specification

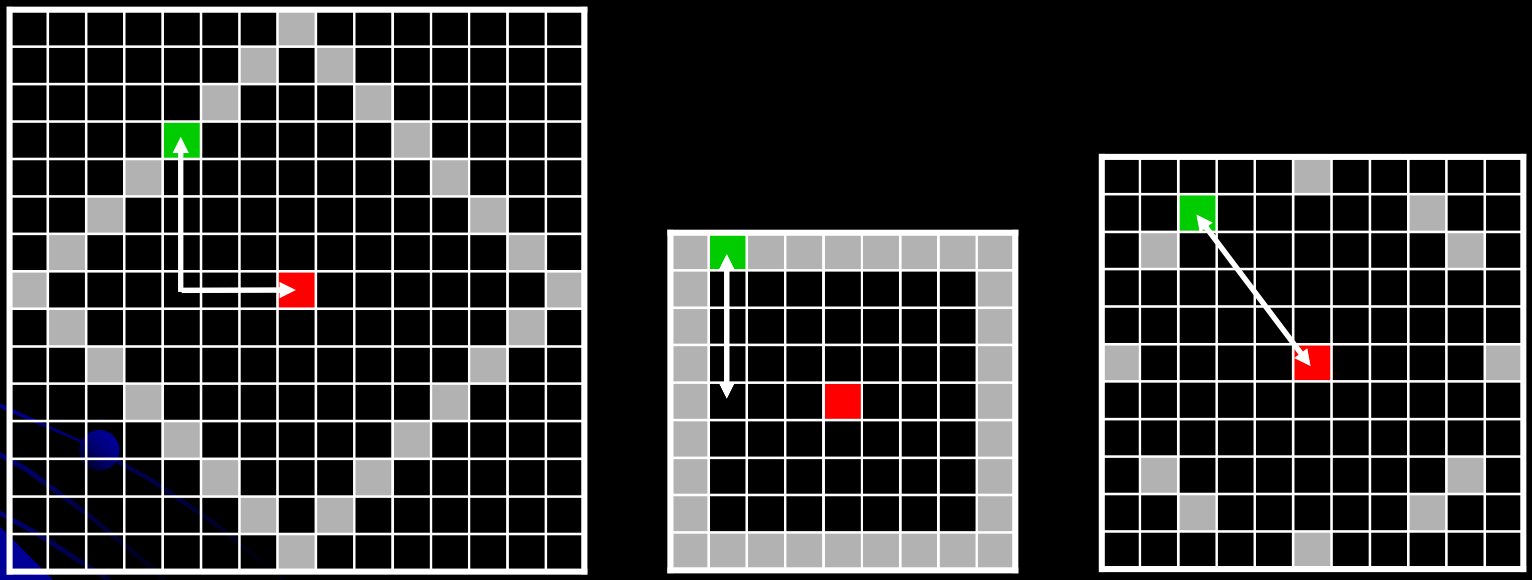
$$O(i, i^*, t) = e^{-\frac{\left\|\mathbf{r}_{i^*}(t) - \mathbf{r}_i(t)\right\|^2}{2\sigma^2(t)}}$$

$$\sigma(t) = \sigma_0 \; e^{\frac{-t}{T_{max}}}$$

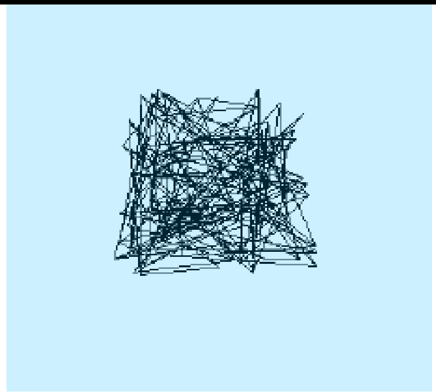




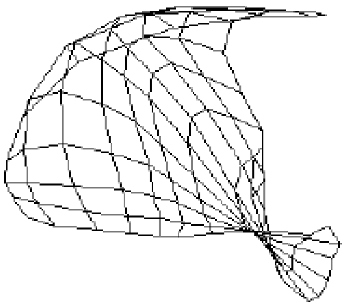
Neighborhood specification



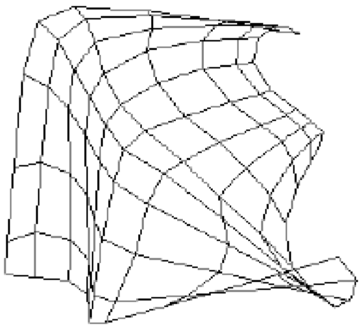
SOM progress



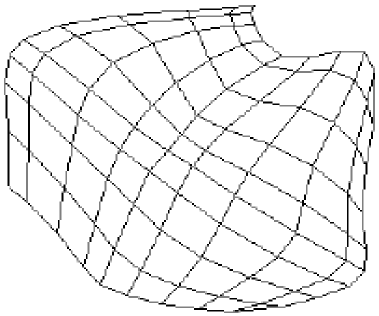
Weights
random numbers from
⟨−0.5,0.5⟩×⟨− 0.5,0.5⟩



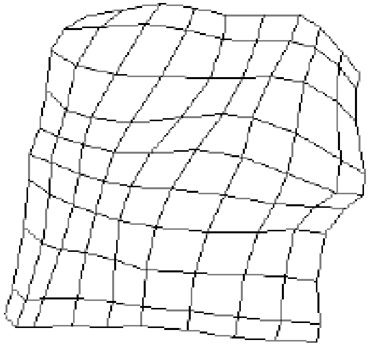
Weights
After 100 iterations



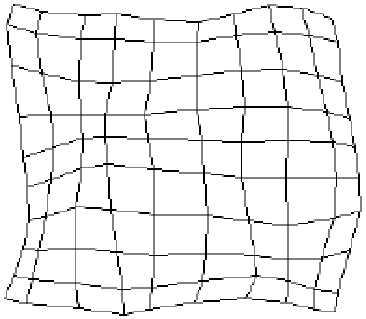
Weights
After 200 iterations



Weights
After 600 iterations



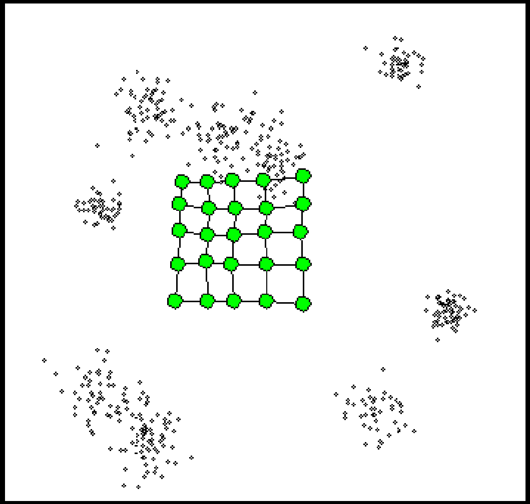
Weights
After 3000 iterations

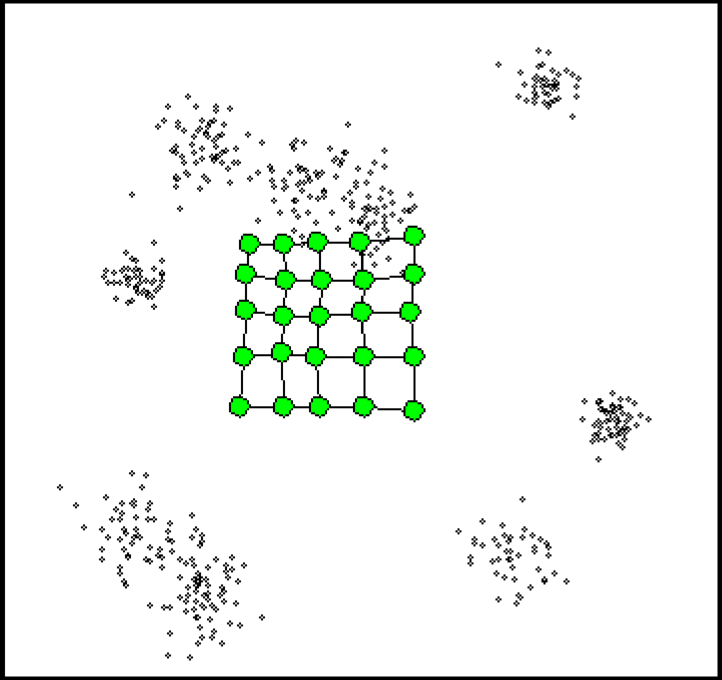


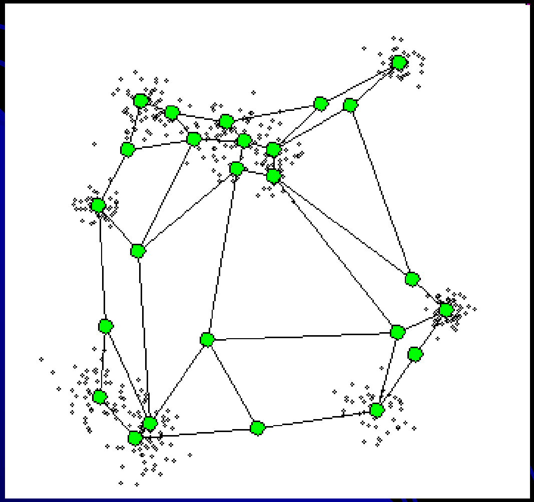
Weights
After 7000 iterations

Input vectors: Uniform random numbers from ⟨−1,1⟩×⟨−1,1⟩.

SOM progress

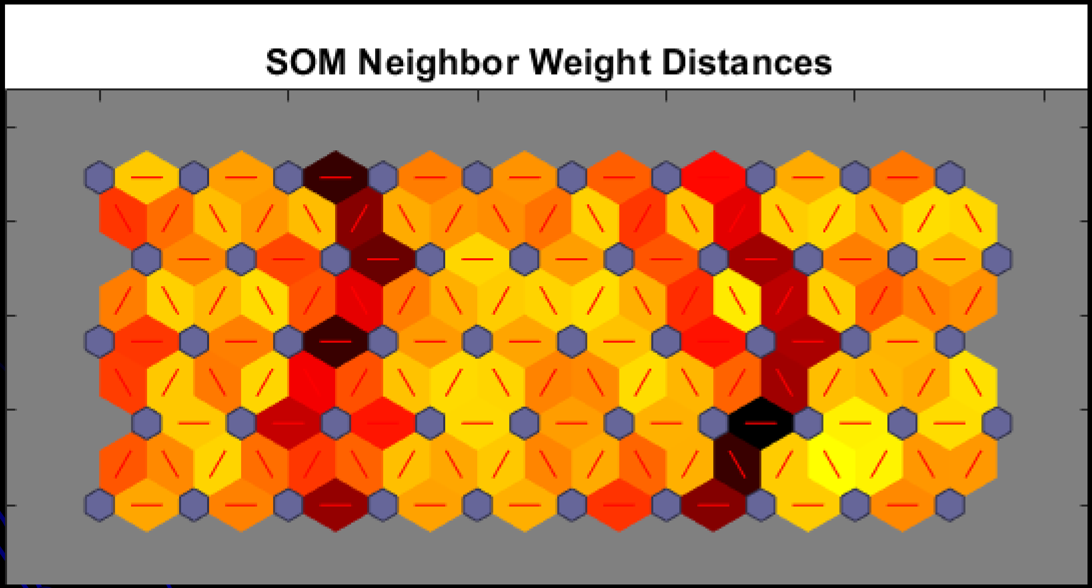


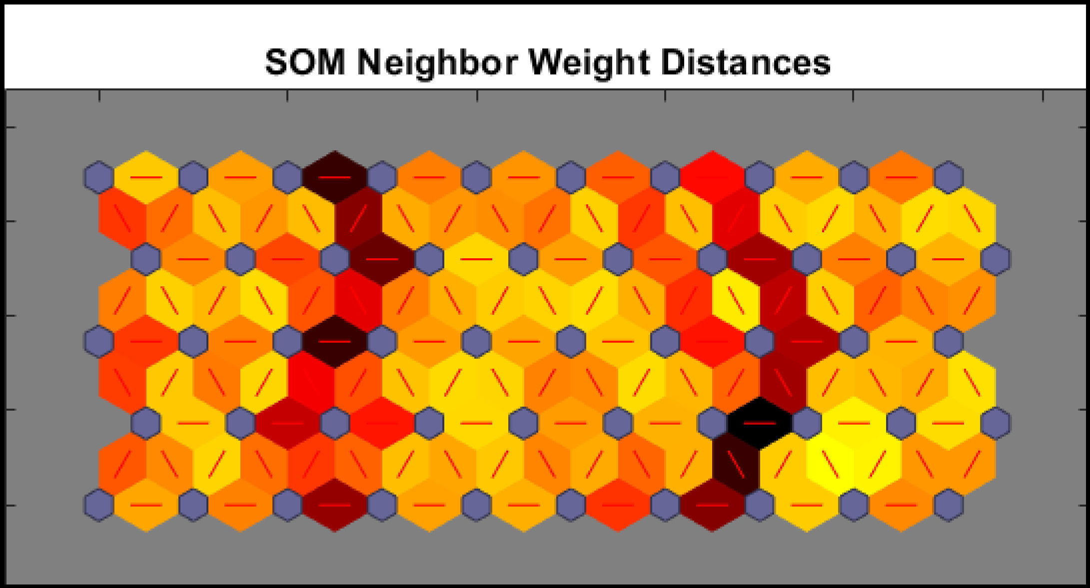


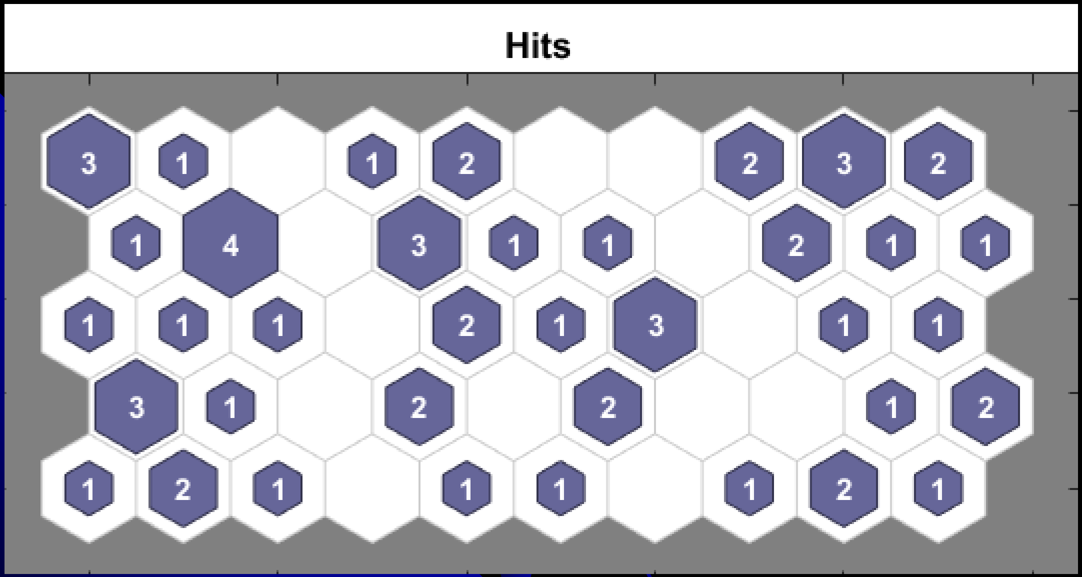


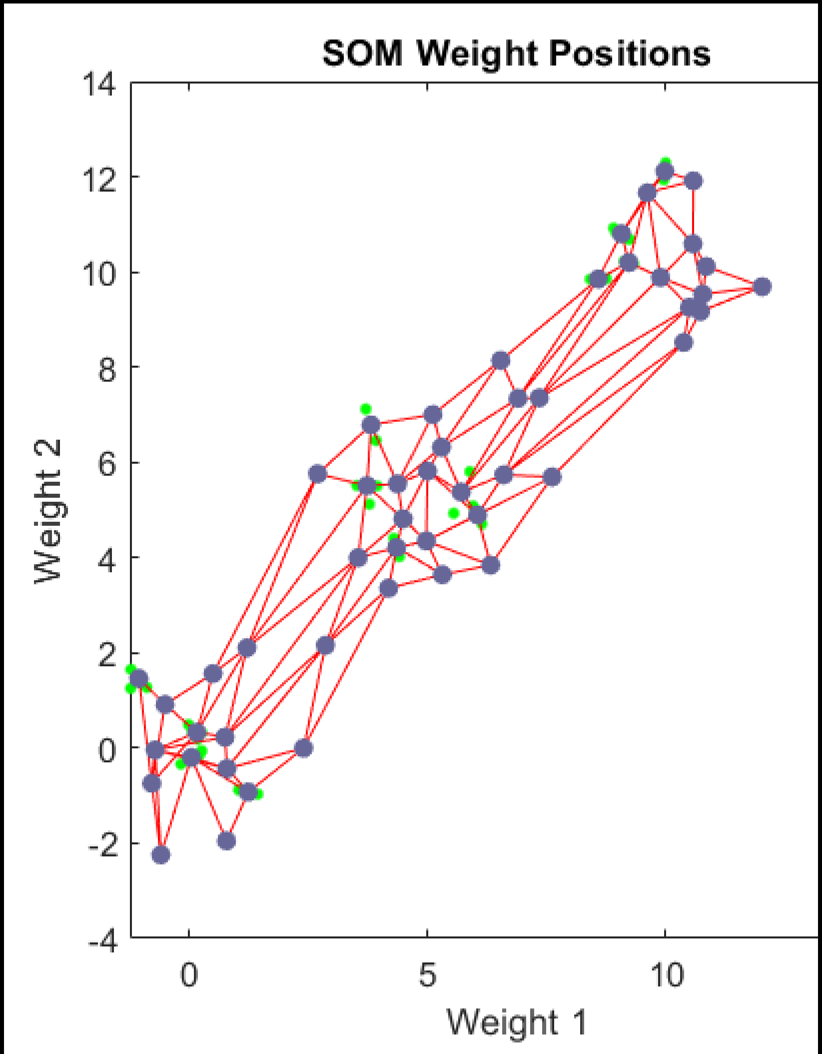
Classification

U-matrix (unified distance matrix):
visualizes the distances between the neurons

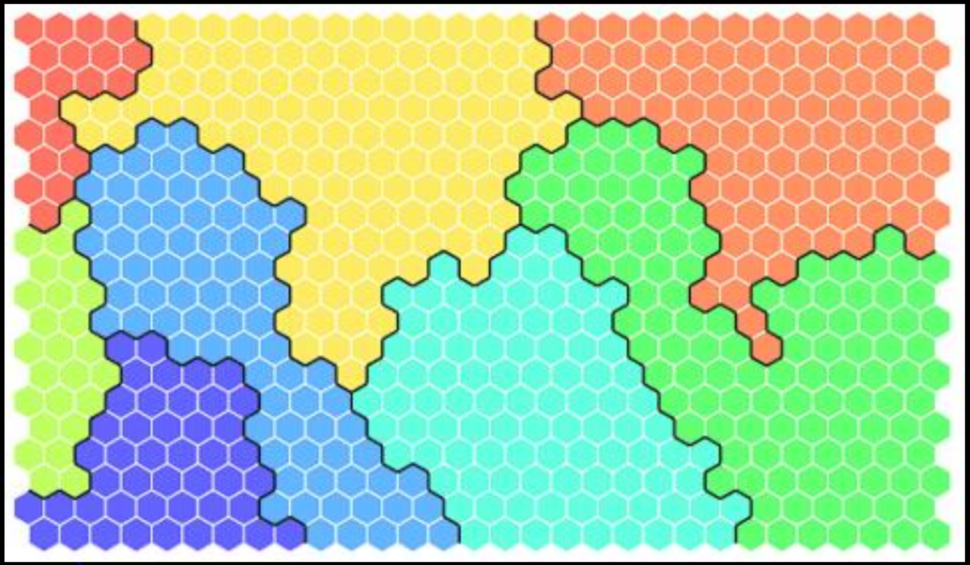



SOM Neighbor Weight Distances


Hits
3
1
1
2
2
3
2
1
4
3
1
1
2
1
1
1
1
1
2
1
3
1
1
3
1
2
2
1
2
1
2
1
1
1
1
2
1


SOM Weight Positions
Weight 2
Weight 1
14
12
10
8
6
4
2
0
-2
-4
0
5
10

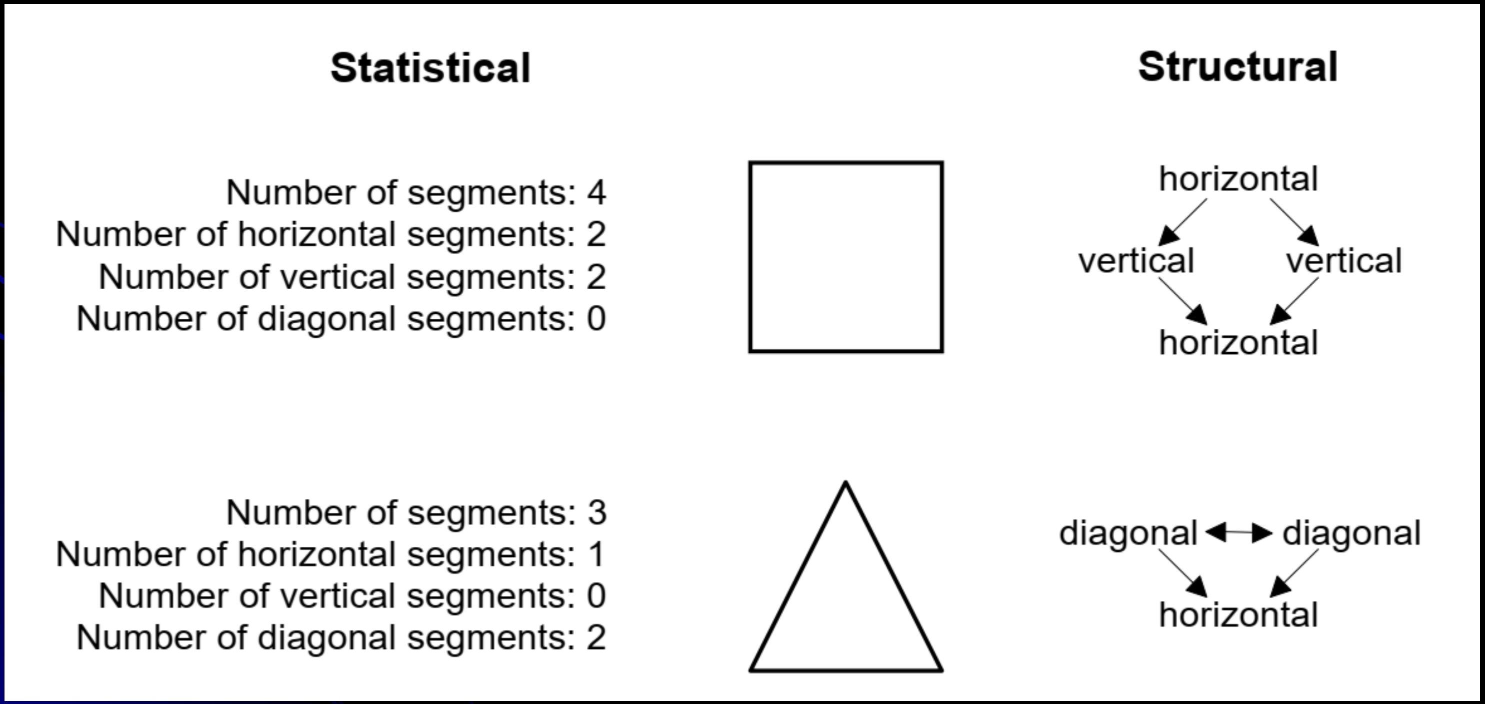
Classification



Structural (syntactic) recognition

Structural pattern recognition

Patterns can contain structural and relational information that are difficult or impossible to quantify in feature vector form



Structural pattern recognition

- Structure quantification and description are mainly done using:
  - Formal grammars
  - Relational descriptions (principally graphs)
- Recognition and classification are done using:
  - Parsing (for formal grammars)
  - Relational graph matching (for relational descriptions)

Applications

a) Classification of time data (e.g. ECG)

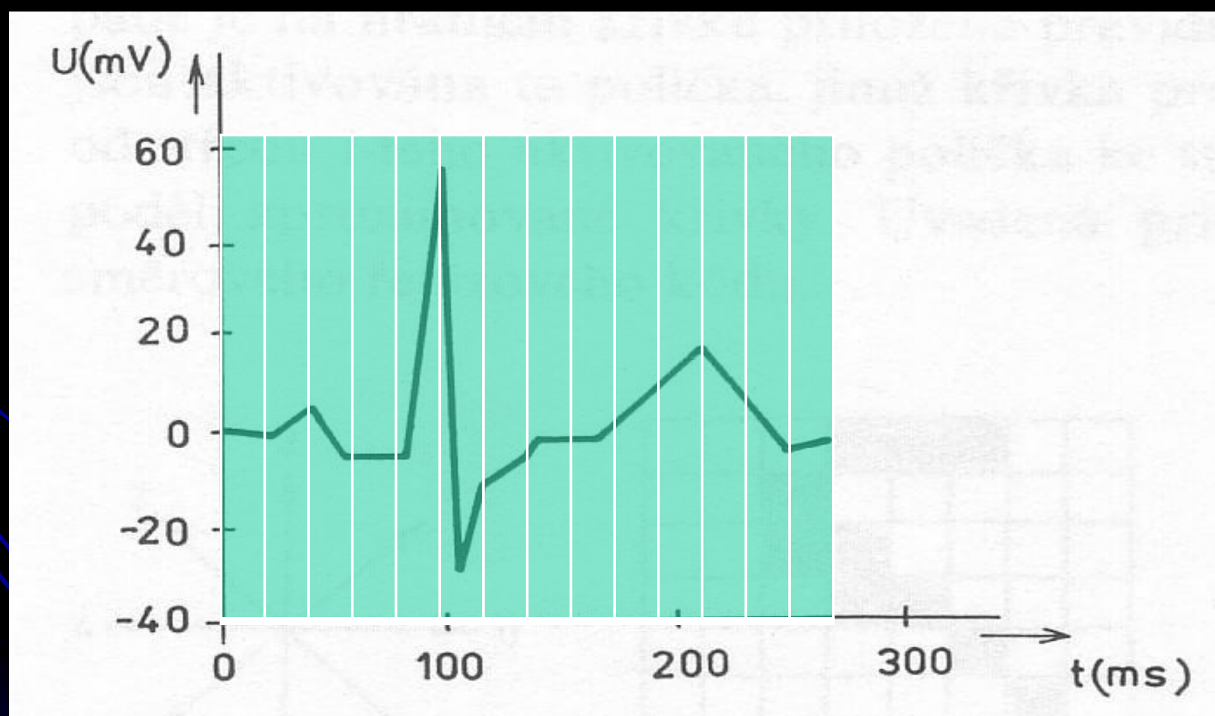
b) Object recognition described by structural codes (e.g. Freeman code, signature…)

c) Scene recognition, scene represented as a graph of primitive objects

Time data

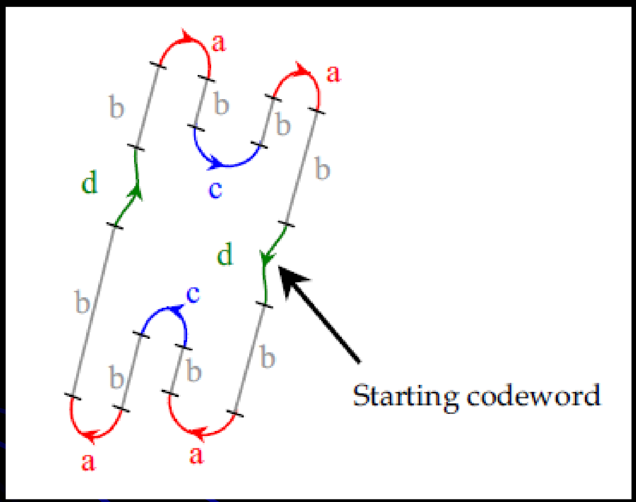
Line approximation of ECG:

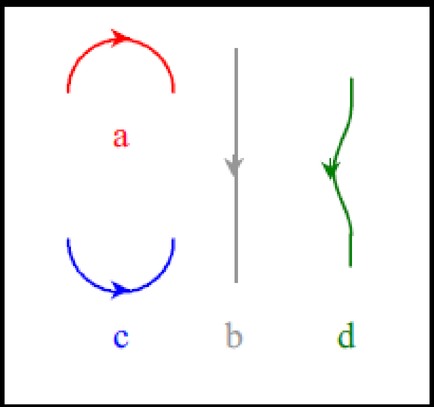
0 / \ 0 / \ / 00 / / \ \ 0



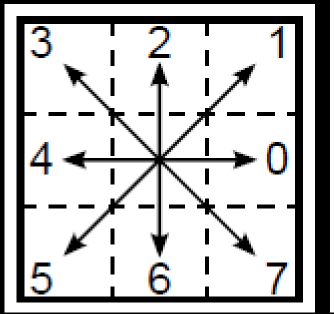
Structural description of objects

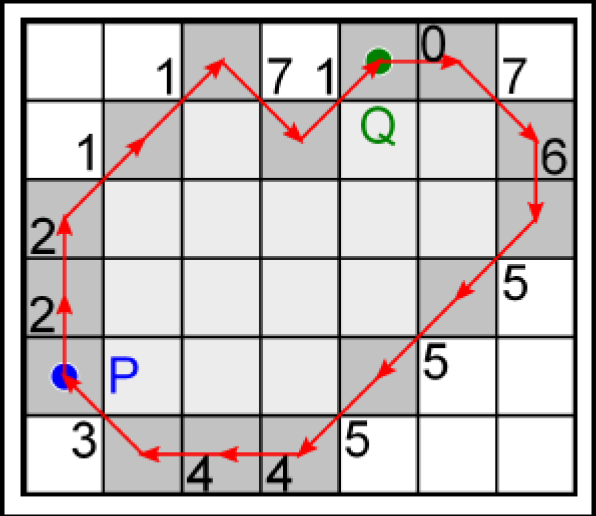
d, b, a, b, c, b, a, b, d, b, a, b, c, b, a, b





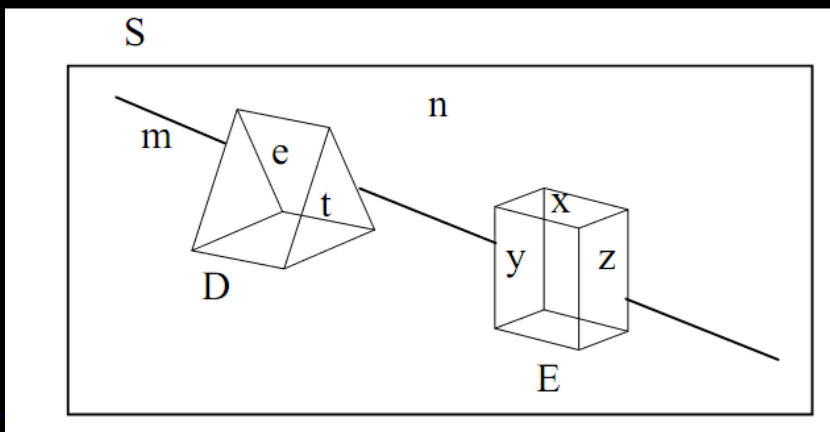
2 2 1 1 7 1 0 7 6 5 5 5 4 4 3





Structural scene description

Hierarchical tree structure



scene S

objects B background C

object D object E floor m wall n

sfc e sfc t sfc x sfc y sfc z

Recognition

Theory of formal languages

A grammar generates a (possibly infinite) set of strings (objects)

If we can design a grammar which generates a class of strings, then we can build a machine which will recognize any string in that class

Formal languages

Alphabet is a finite set of symbols, $V=\{x_1,x_2, \ldots,x_n\}$

Word over V is a finite string of ordered symbols from V

Example: V = {a,b,c}, valid words are “abcab”, “abba”, “aaa”, null

V* set of all words over V

Language is an arbitrary subset L of V*

Example: V={0,1}, then L_1 = {001, 110, 111, 0, null} is a finite language

$L_2 = \{s \mid s = 1^n0^21^m, n>=1, 1<=m<=10\}$ is an infinite language

Recognition

Objects from one class – words from the language of this class

Classification – decide whether a word belongs to the language of a class

Finite language – check all words

Infinite language – use the language grammar or automaton to check

Grammars

Grammar G = {V_T, V_N, P, S}

- V_T is a set of terminal symbols
- V_N is a set of *non-terminal* symbols
- $V_T \cap V_N = \varnothing$;
- P is the set of production rules $(V_T \cup V_N)^* V_N (V_T \cup V_N)^* \rightarrow (V_T \cup V_N)^*$
- S is the starting symbol or the root; S belongs to V_N

L(G) is a formal language generated by the grammar G

- Each string is composed of only terminals
- Each string can be derived from S using the production rules P
- Example: V_T = {a,b}, V_N = {S}; P = {S->aSb, S->ab} => L(G) : $a^n b^n$, n>=1

Inference

Derive grammar based on training set or domain knowledge

Not unique solutions

No general method, usually user interaction is required

Example

Consider,

a: 0^o horizontal unit length

b: 120^o unit length

c: 240^o unit length

$$L = \{a^n b^n c^n ; 1 \leq n \leq 3\}$$

L(G) represents the class of equilateral triangles

What is the grammar?

Example

Type 3 Grammar solution

V_T = {a,b,c}

V_N = {S, A, B, C, D, E, F, G, H, I, J, K}

$$\begin{array}{lll} S \mapsto aA & C \mapsto bI & H \mapsto bK \\ S \mapsto aC & D \mapsto bF & I \mapsto c \\ A \mapsto aB & F \mapsto bJ & J \mapsto cI \\ A \mapsto aD & E \mapsto bG & K \mapsto cJ \\ B \mapsto aE & G \mapsto bH & \end{array}$$

Type 2 Grammar solution

V_T = {a,b,c}

V_N = {S, A, B, C, D, E, F}

$$\begin{array}{ccc} S \mapsto aAF & A \mapsto aBF & D \mapsto bc \\ A \mapsto b & B \mapsto aEF & C \mapsto b \\ A \mapsto aDF & E \mapsto bD & F \mapsto c \end{array}$$

Inference from training set

input: T = {x1,..., xt}

output: regular grammar G = (VN, VT, S, P)

Step 1

Find all terminals in T → create VT

Step 2

For each word xi = ai1...ain (xi∈T) create rules

S → ai1Z i1

Zi1 →ai2Zi2

.....

Zi,n-2 →ai,n-1Zi,n-1

Zi,n-1 →ain

every Zij is a new non-terminal

Example

Regular grammar G* .... unknown

G* = ({S,A,B,C}, {a,b}, S, P)

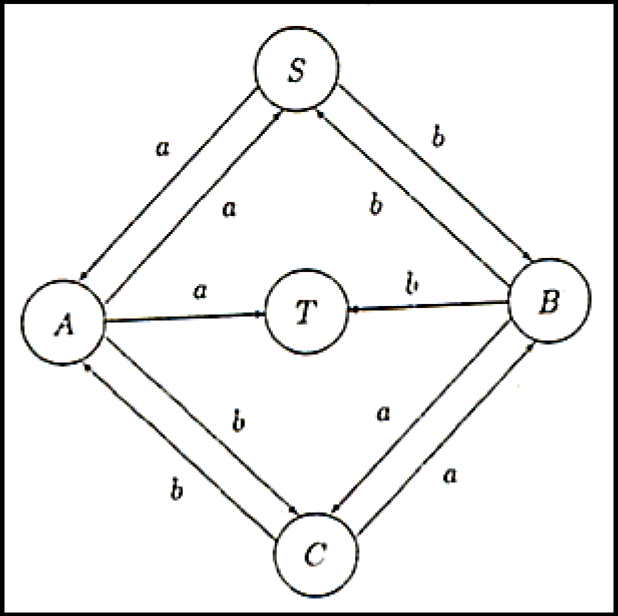
S → aA | bB

A → a | aS | bC

B → b | bS | aC

C → aB | bA

Finite automaton of G*



Training set

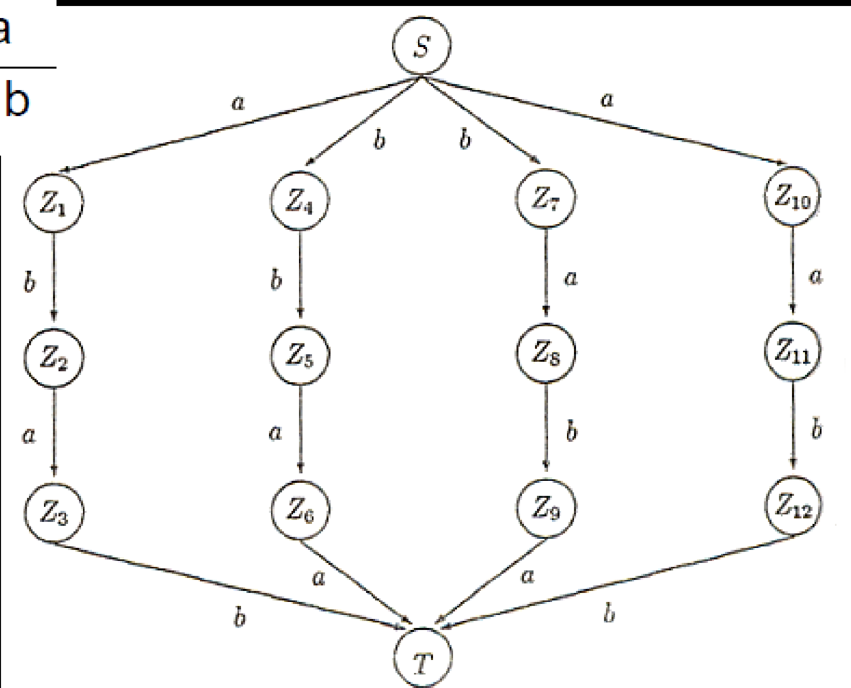
T = {abab, bbaa, baba, aabb}

Example

Inference: VT = {a,b}

VN = $\{S, Z_1, Z_2, Z_3, Z_4, Z_5, Z_6, Z_7, Z_8, Z_9, Z_{10}, Z_{11}, Z_{12}\}$

$S \to aZ_1$	$Z_1 \to bZ_2$	$Z_2 \to aZ_3$	$Z_3 \to b$
$S \to bZ_4$	$Z_4 \to bZ_5$	$Z_5 \to aZ_6$	$Z_6 \to a$
$S \to bZ_7$	$Z_7 \to aZ_8$	$Z_8 \to bZ_9$	$Z_9 \to a$
$S \to aZ_{10}$	$Z_{10} \to aZ_{11}$	$Z_{11} \to bZ_{12}$	$Z_{12} \to b$



Training set
T = {abab, bbaa, baba, aabb}

Example

L(G*) – infinite

L(G) – finite

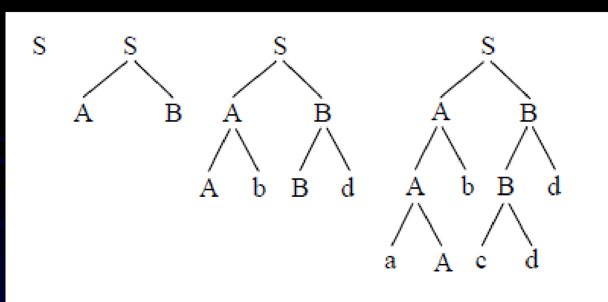
$L(G) = T \subseteq L(G^*)$

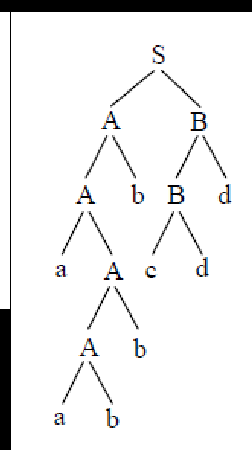
Many non-terminals, some equivalent

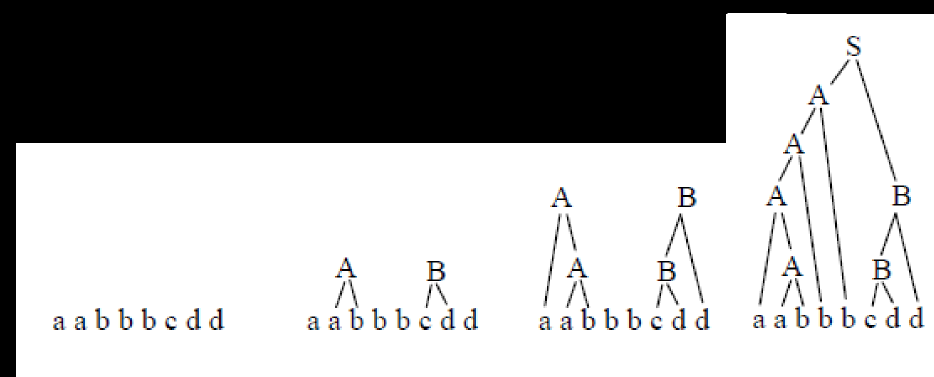
Generates only words from the training set

Recognition

- 1st step – check the terminal symbols
- 2nd step – try to derive the word from compliant grammars: top-down, bottom-up







Syntactic deformations

Errors, noise, …

Structural deformations

Search for most similar word

e.g. Levenshtein distance (number of transformations needed to transform word A to B)

Grammars contains deformation rules (insertion, deletion, substitution)

Summary

- The classifier for a structural pattern recognition system consists of a set of grammars, one for each class
- The main difficulty lies in grammar inference
- Applications – mainly user-constructed grammars


END OF TERM:
FREEDOOOOOOM!!!
makeameme.org