

Unsupervised Learning

CS 445/545

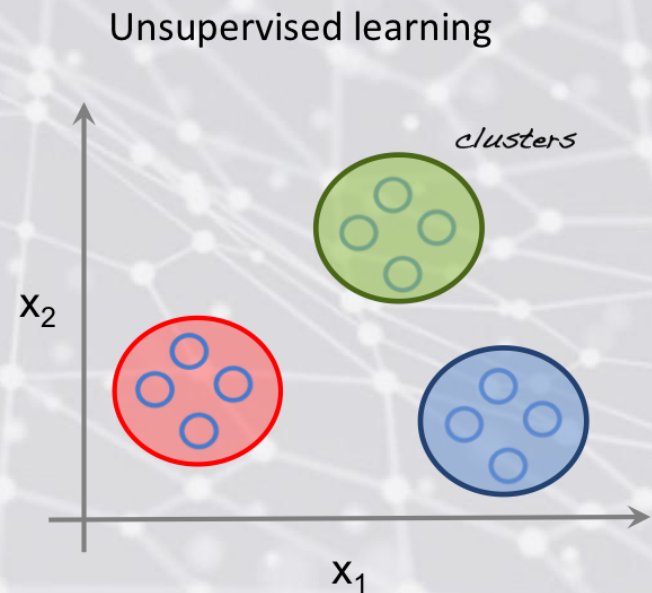
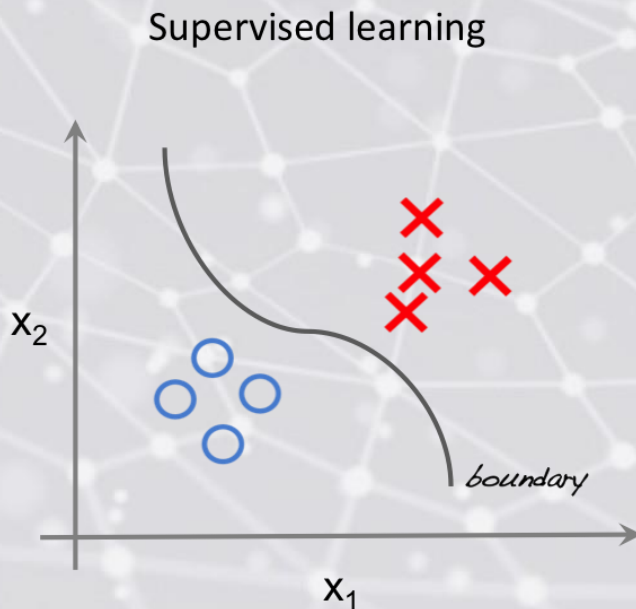


Outline

- Overview of unsupervised learning
- K-means/fuzzy c-means
- Gaussian Mixture Models (*GMMs*)
- Cluster Analysis
- Hierarchical Clustering
- *DBSCAN*
- Vector Quantization / Self-organizing maps (SOMs)

Overview

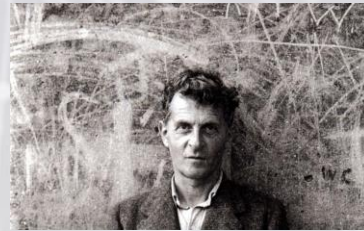
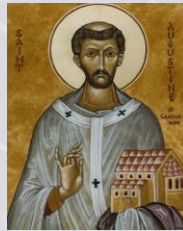
- To date, most of the learning algorithms that we have seen make use of **labelled training data** (*e.g.* a discrete class assignment or real-valued targets).
- Target labels are useful, since we can use them to guide the learning process (see: PLA, backprop, etc.). However, in many real-world circumstances labelled data is difficult (and expensive) to obtain.



Introduction: Perspectives on Learning

- In addition, labelled data is not seemingly biologically plausible – when we are learning we rarely encounter labelled examples. *Or do we?*
- Here are (2) contrasting perspectives on epistemology and learning theory from the history of philosophy.

“When they (my elders) named some object, and accordingly moved towards something, I saw this and I grasped that the thing was called by the sound they uttered when they meant to point it out. Their intention was shewn by their bodily movements, as it were the natural language of all peoples: the expression of the face, the play of the eyes, the movement of other parts of the body, and the tone of voice which expresses our state of mind in seeking, having, rejecting, or avoiding something. Thus, as I heard words repeatedly used in their proper places in various sentences, I gradually learnt to understand what objects they signified; and after I had trained my mouth to form these signs, I used them to express my own desires.” -- Augustine, *Confessions*.



“[T]he term ‘language-game’ is meant to bring into prominence the fact that the speaking of language is part of an activity, or a form of life ... [M]eaning can be defined thus: the meaning of word is its use in the language ... And now, I think we can say: Augustine [only] describes the learning of human language as if a child did not understand the language and came into a strange country and did not understand the language of the country; that is, as if it already had a language, only not this one.” –Ludwig Wittgenstein, *Philosophical Investigations*.

* Recommended recreational reading: Augustine, *Confessions*; Wittgenstein, *Philosophical Investigations*.



Are Categories Fundamentally Ambiguous?

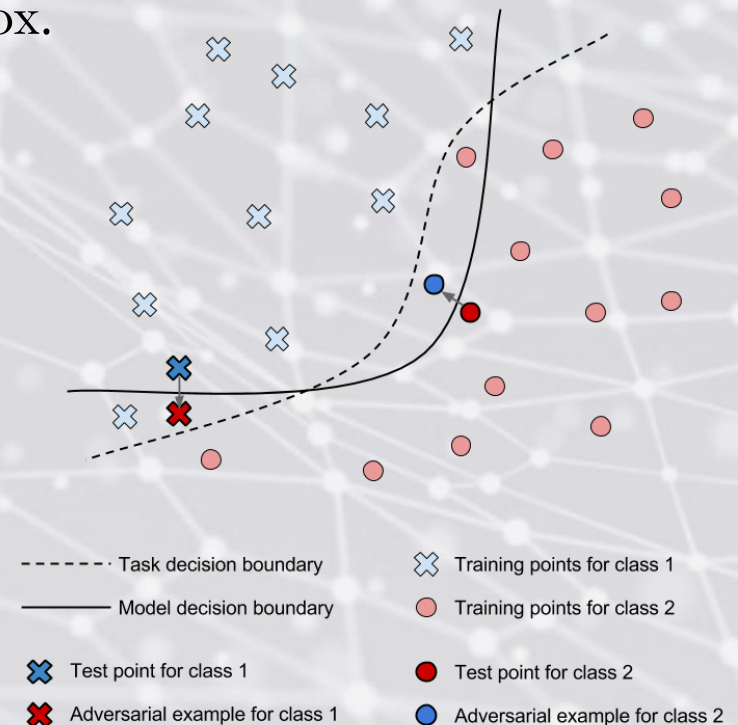
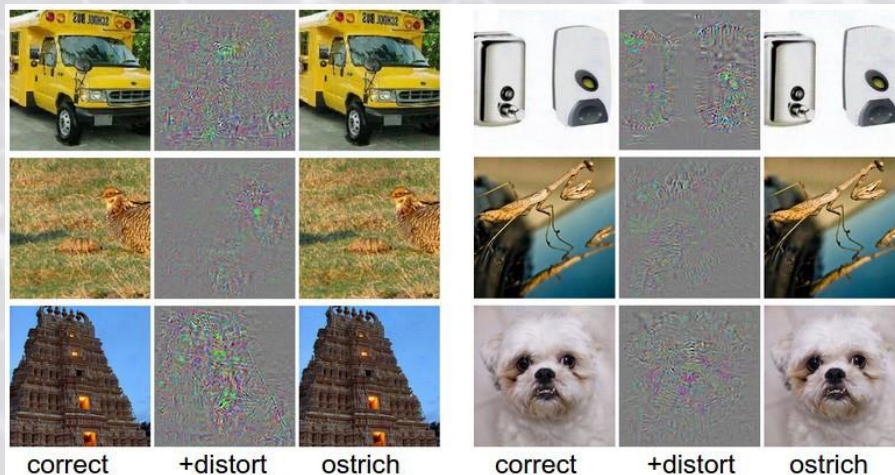
- Consider the ancient thought experiment, *Theseus' ship paradox*.

The ship wherein Theseus and the youth of Athens returned from Crete had thirty oars, and was preserved by the Athenians down even to the time of Demetrius Phalereus, for they took away the old planks as they decayed, putting in new and stronger timber in their places, in so much that this ship became a standing example among the philosophers, for the logical question of things that grow; one side holding that the ship remained the same, and the other contending that it was not the same. — Plutarch

Plutarch thus questions whether the ship would remain the same if it were entirely replaced, piece by piece.

Introduction

- Recall the previously referenced paper from Goodfellow *et al* (2016) on GANs (generative adversarial networks) that problematized the intuitive notion of classification in high dimensions.
- This seminal paper (among other related papers) rightfully complicated our current understanding of deep learning and DNNs – in a way, it conjures up a modern variant of the Theseus ship paradox.



Introduction

- Unsupervised learning is a conceptually different problem from supervised learning – no target labels are available to us.

Q: How can we hope to perform classification?

Introduction

- Unsupervised learning is a conceptually different problem from supervised learning – no target labels are available to us.

Q: How can we hope to perform classification?

A: Identify similarities between inputs, and consider groups of similar inputs as belonging to the same ***cluster***.

(*) Accordingly, the aim of unsupervised learning is to find clusters (according to some useful criteria) of similar inputs in the data without being told explicitly that these data belong to a particular class.

In this way, an unsupervised learning algorithm “discovers” similarities and patterns in the data itself. This procedure is often part of a more general **exploratory data analysis (EDA)** methodology, common to data science, statistics, etc.

Introduction

- Generally, supervised learning algorithms aim to minimize some *external error criterion* (e.g., OLS), based on a loss function quantifying the difference between targets and outputs.
 - With supervised learning, calculating this loss explicitly was possible because we were provided with target labels.
 - Instead, with unsupervised learning, we need to use an **internal error criterion**. This means that the measure has to be independent of the task – think of the labels as defining a particular task.
- (*) A useful **general error criterion** in the unsupervised setting defines similarity in terms of the distance between data points; accordingly, similar data are close to one another.

Example: Optdigits data set

[illegible][illegible][illegible]

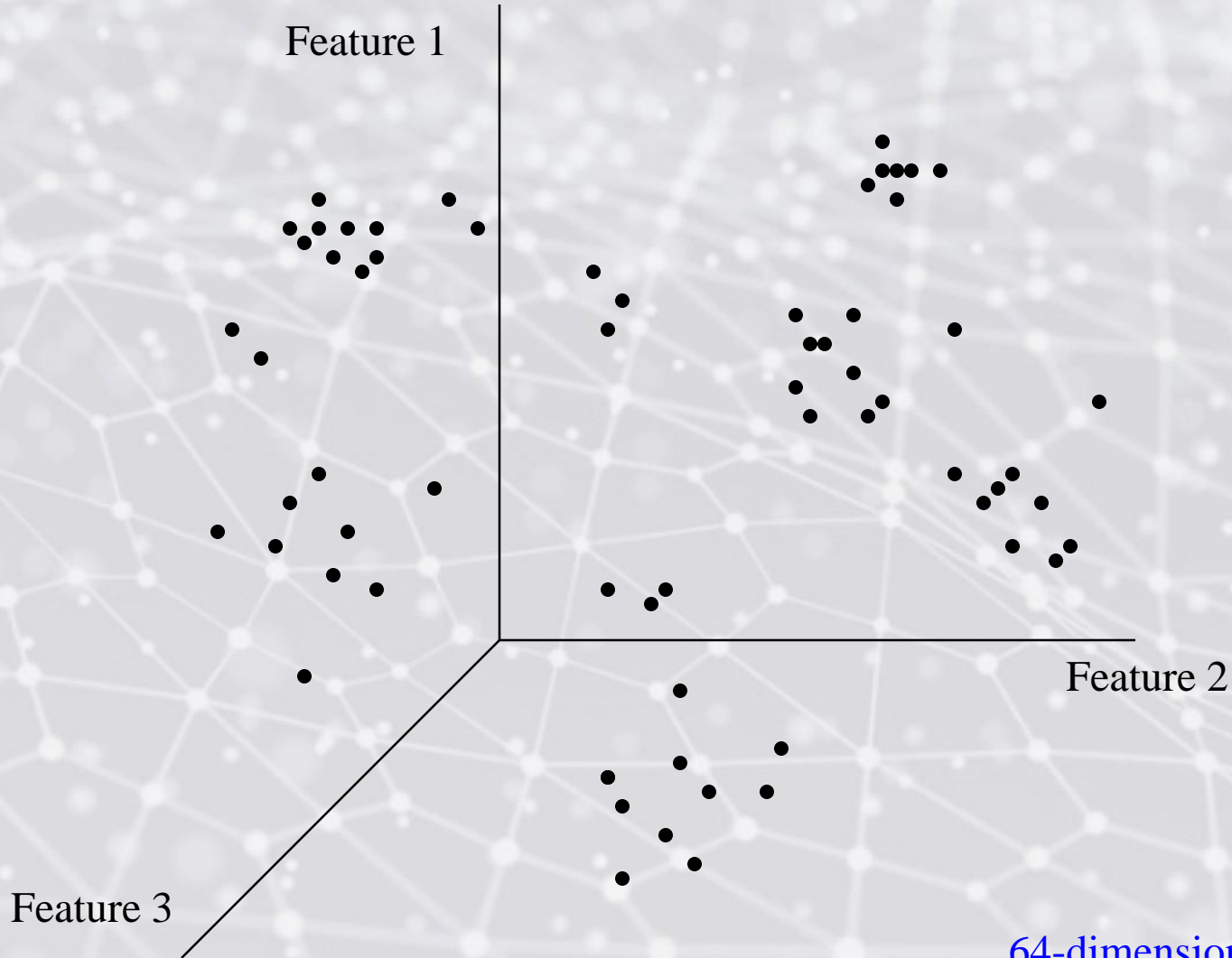
Optdigits features

The diagram illustrates a 2D array structure with 8 columns and 8 rows. The columns are labeled $f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8$ at the top. The rows are labeled f_9 on the left. A blue box highlights the first four rows and the first four columns. The text "Etc. .." is placed in the middle of the grid, indicating the pattern continues.

$$\mathbf{x} = (f_1, f_2, \dots, f_{64})$$

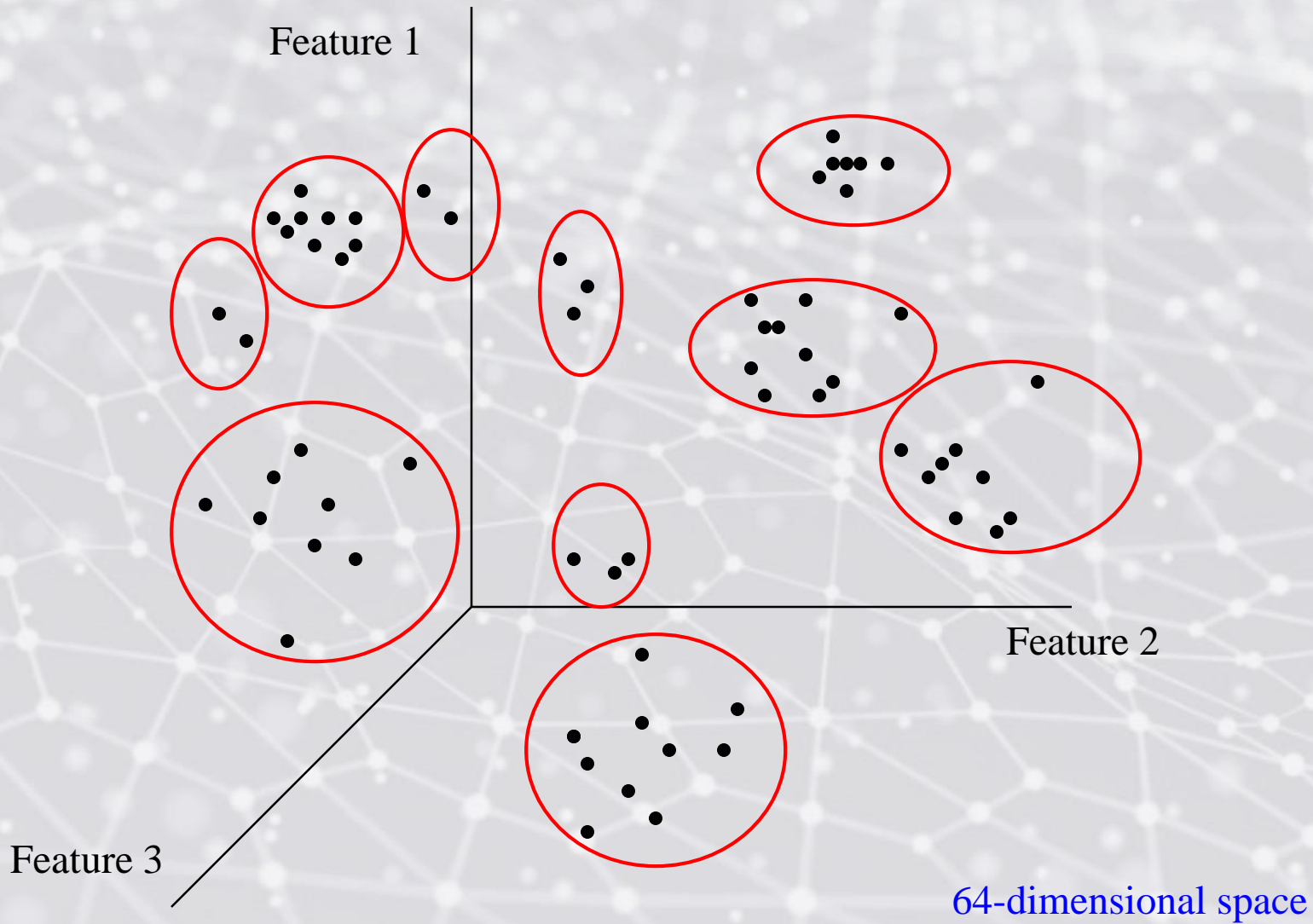
$$= (0, 2, 13, 16, 16, 16, 2, 0, 0, \dots)$$

Partitional Clustering of Optdigits



64-dimensional space

Partitional Clustering of Optdigits

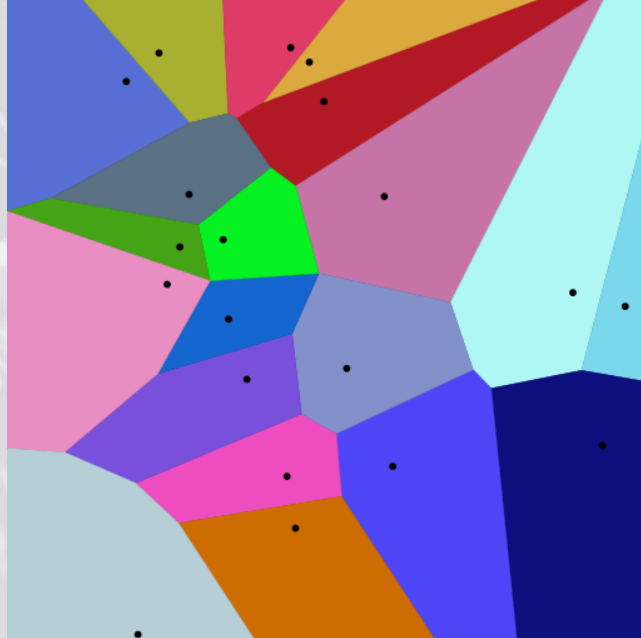


Issues for clustering algorithms

- How to measure distance between pairs of instances?
- How many clusters to create?
- Should clusters be hierarchical? (i.e., clusters of clusters)
- Should clustering be “soft”? (i.e., an instance can belong to different clusters, with “weighted belonging”)

k-Means

- k-means is a very popular (and simple) clustering algorithm used in ML and data science.
- k -means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a *prototype of the cluster*. This results in a partitioning of the data space into **Voronoi cells**.



*Voronoi
Tessellation; 20
points and their
Voronoi cells.*



k-Means

- Given a set of observations $(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)$, where each observation is a d-dimensional real vector, k-means clustering endeavors to partition the n observations into k ($\leq n$) sets $\mathbf{S} = \{S_1, S_2, \dots, S_k\}$ so as to **minimize the within-cluster sum of squares (WCSS)**.
- Formally, the objective is to find:

$$\arg \min_{\mathbf{S}} \sum_{i=1}^k \sum_{x \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2 = \arg \min_{\mathbf{S}} \sum_{x \in S_i} |S_i| \text{Var}(S_i)$$

where $\boldsymbol{\mu}_i$ is the mean of cluster S_i .

k-Means

- The algorithm itself works by iterative refinement, and is a variant of a more general algorithm, known as **EM** (**expectation-maximization**).
- Given an initial set of k means $m_1^{(1)}, \dots, m_k^{(1)}$ (the subscript is the cluster identification, while superscript is the iteration number) k-means alternates between the following (2) steps:

(I) Assignment Step (i.e., the expectation step):

Assign each observation to the cluster whose mean has the least squared Euclidean distance, this is intuitively the "nearest" mean. Mathematically, this means partitioning the observations according to the Voroni tessellation generated by the means.

$$S_i^{(t)} = \left\{ x_p : \|x_p - m_i^{(t)}\|^2 \leq \|x_p - m_j^{(t)}\|^2 \quad \forall j, 1 \leq j \leq k \right\}$$

Where each datum x_p is assigned to exactly one cluster, $S^{(t)}$.

k-Means

- Given an initial set of k means $m_1^{(1)}, \dots, m_k^{(1)}$ k-means alternates between the following (2) steps:

(I) Assignment Step (i.e., the expectation step):

Assign each observation to the cluster whose mean has the least squared Euclidean distance, this is intuitively the "nearest" mean. Mathematically, this means partitioning the observations according to the Voroni tessellation generated by the means.

$$S_i^{(t)} = \left\{ x_p : \|x_p - m_i^{(t)}\|^2 \leq \|x_p - m_j^{(t)}\|^2 \quad \forall j, 1 \leq j \leq k \right\}$$

(II) Update Step (i.e., the parameter maximization step):

- Calculate the new means to be the **centroids** of the observations in the new clusters.

$$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j$$

- The algorithm has **converged** when the assignments no longer change.
There is no guarantee that the optimum is found using this algorithm.

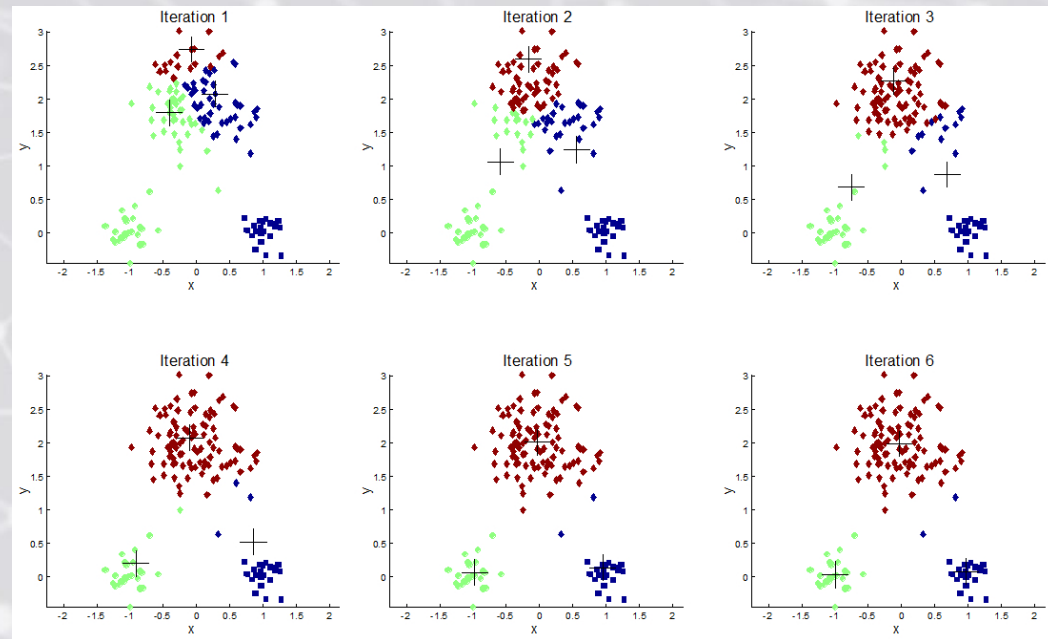
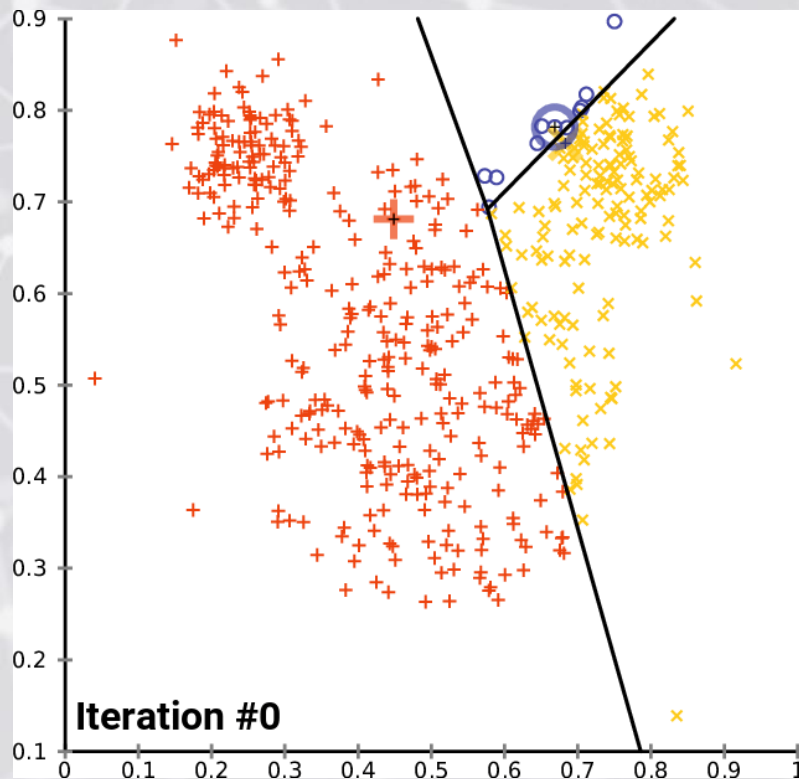
k-Means

(I) **Assignment Step** (i.e., the expectation step):

$$S_i^{(t)} = \left\{ x_p : \left\| x_p - m_i^{(t)} \right\|^2 \leq \left\| x_p - m_j^{(t)} \right\|^2 \quad \forall j, 1 \leq j \leq k \right\}$$

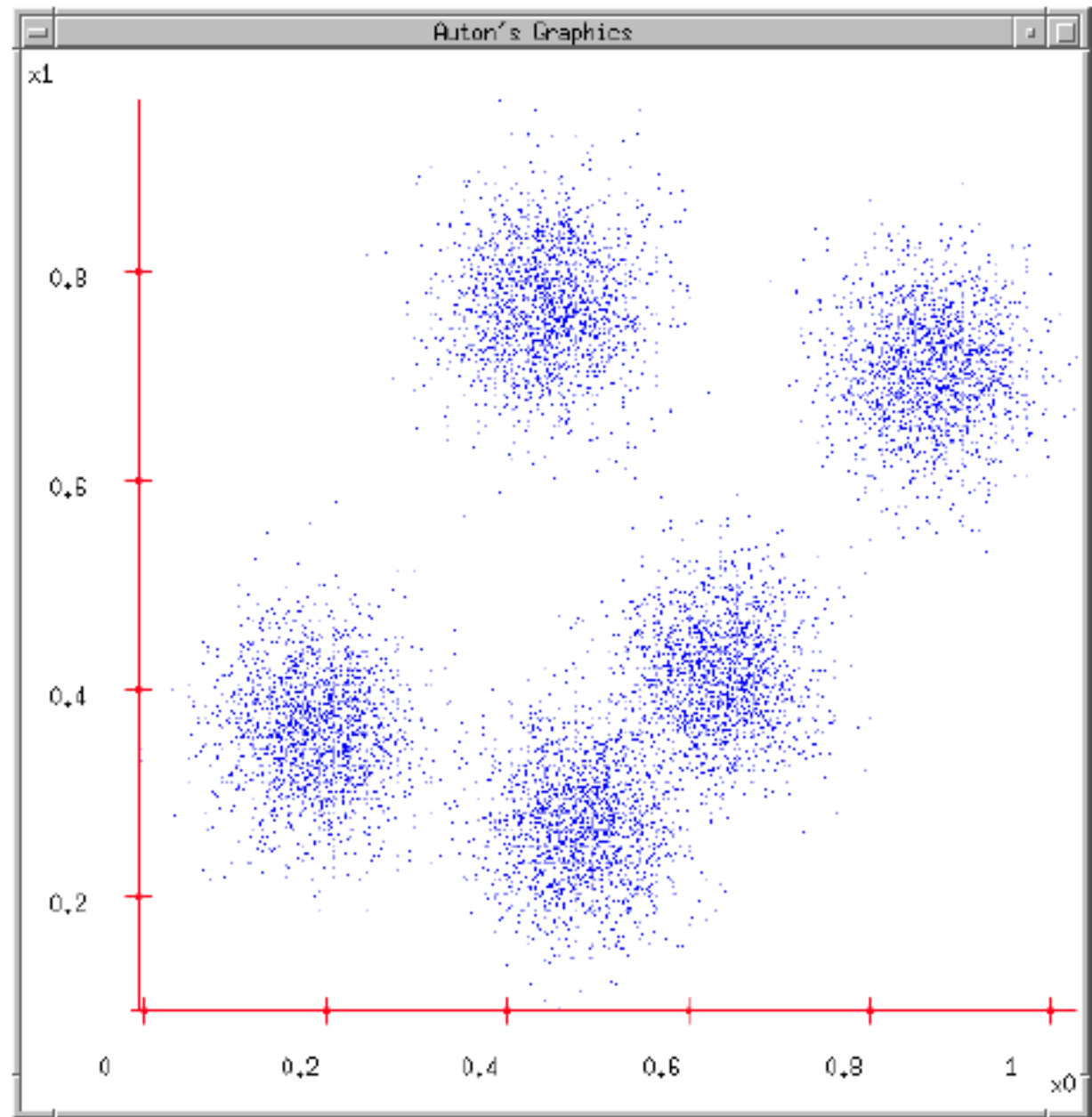
(II) **Update Step** (i.e., the parameter maximization step):

$$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j$$



K-means

1. Ask user how many clusters they'd like.
(e.g. $k=5$)



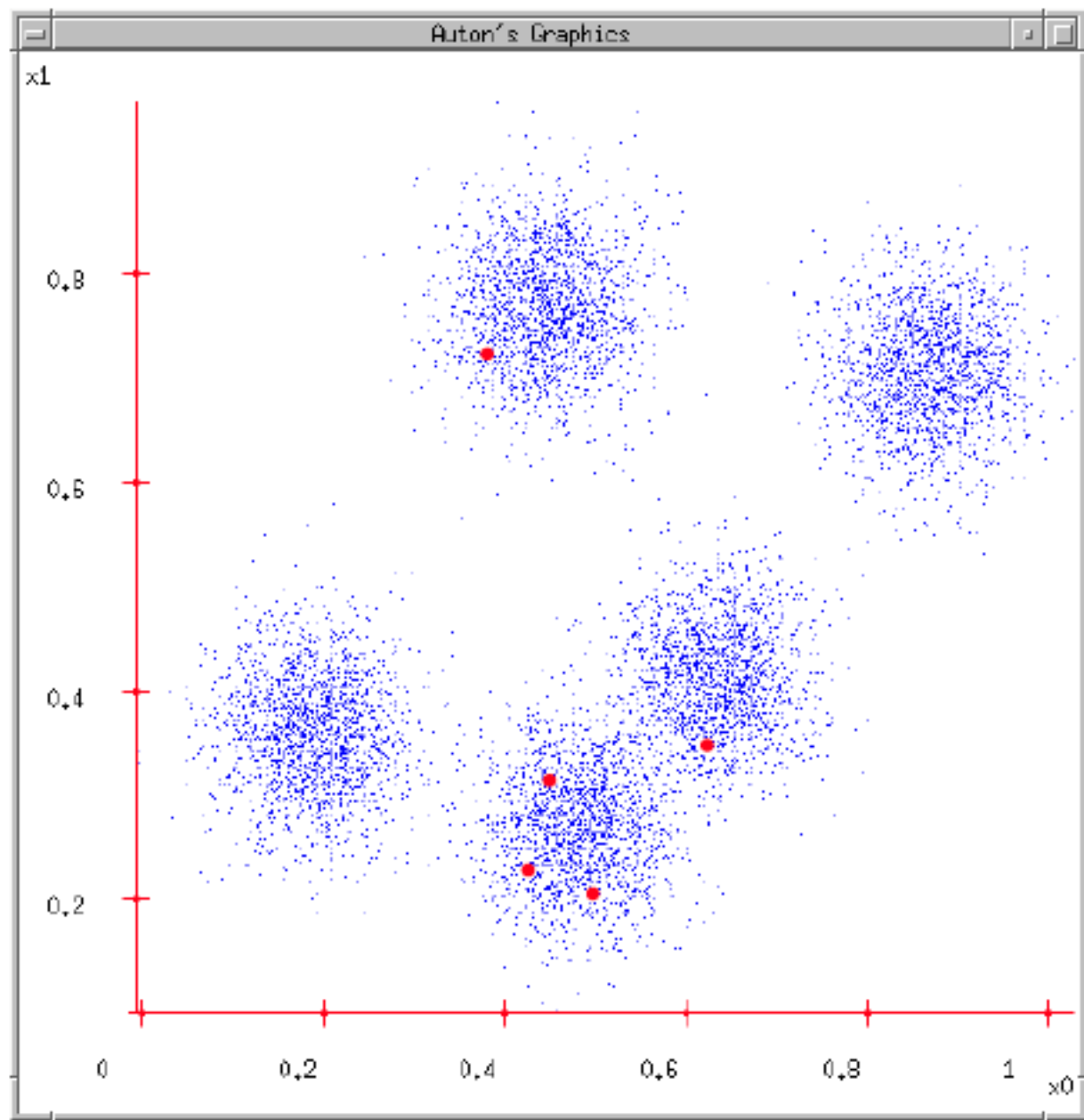
Adapted from Andrew Moore,
<http://www.cs.cmu.edu/~awm/tutorials>

Copyright © 2001, 2004, Andrew W. Moore

K-means and Hierarchical Clustering: Slide 6

K-means

1. Ask user how many clusters they'd like.
(e.g. $k=5$)
2. Randomly guess k cluster Center locations



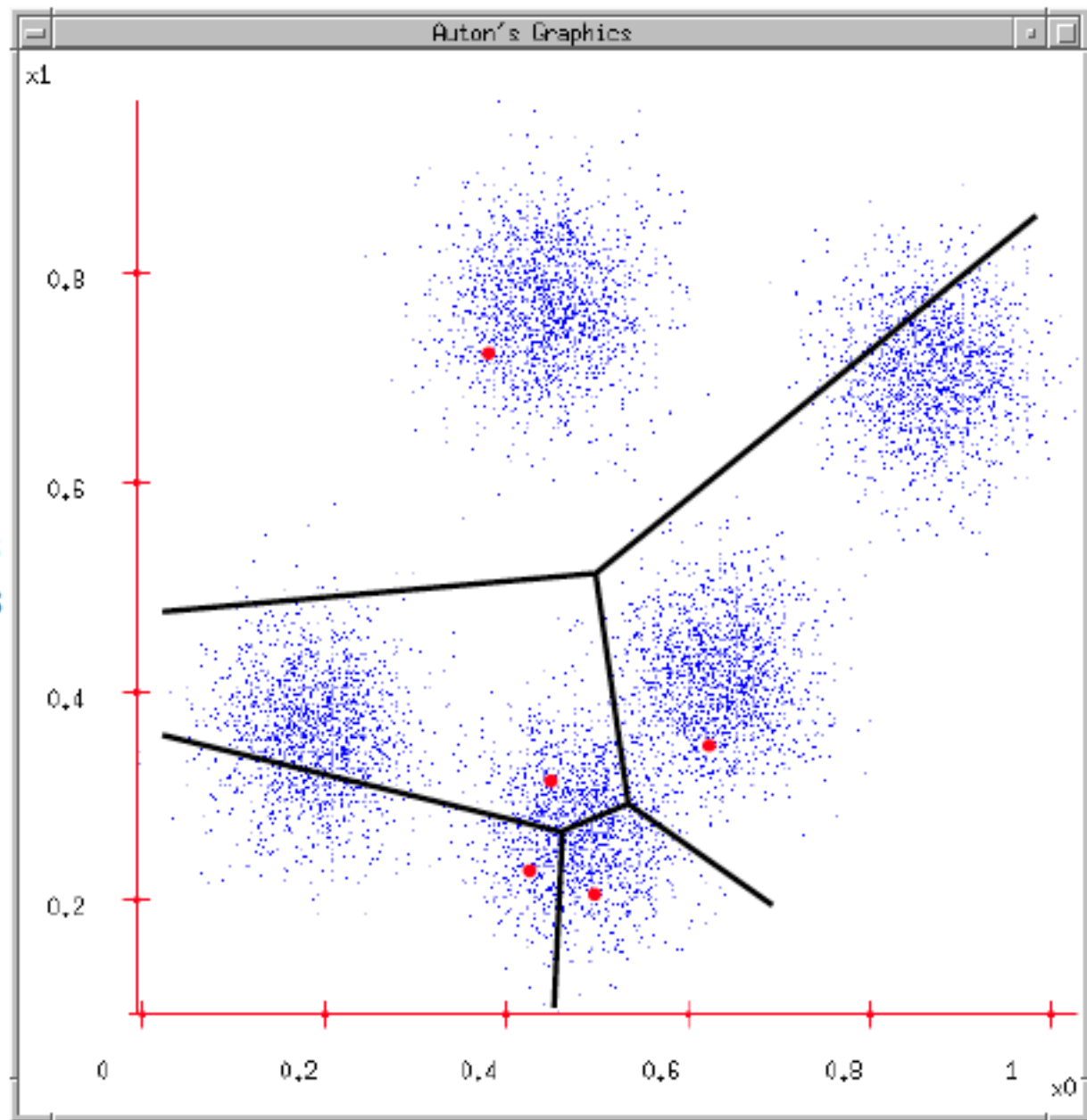
Adapted from Andrew Moore,
<http://www.cs.cmu.edu/~awm/tutorials>

Copyright © 2001, 2004, Andrew W. Moore

K-means and Hierarchical Clustering: Slide 7

K-means

1. Ask user how many clusters they'd like.
(e.g. $k=5$)
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to. (Thus each Center "owns" a set of datapoints)



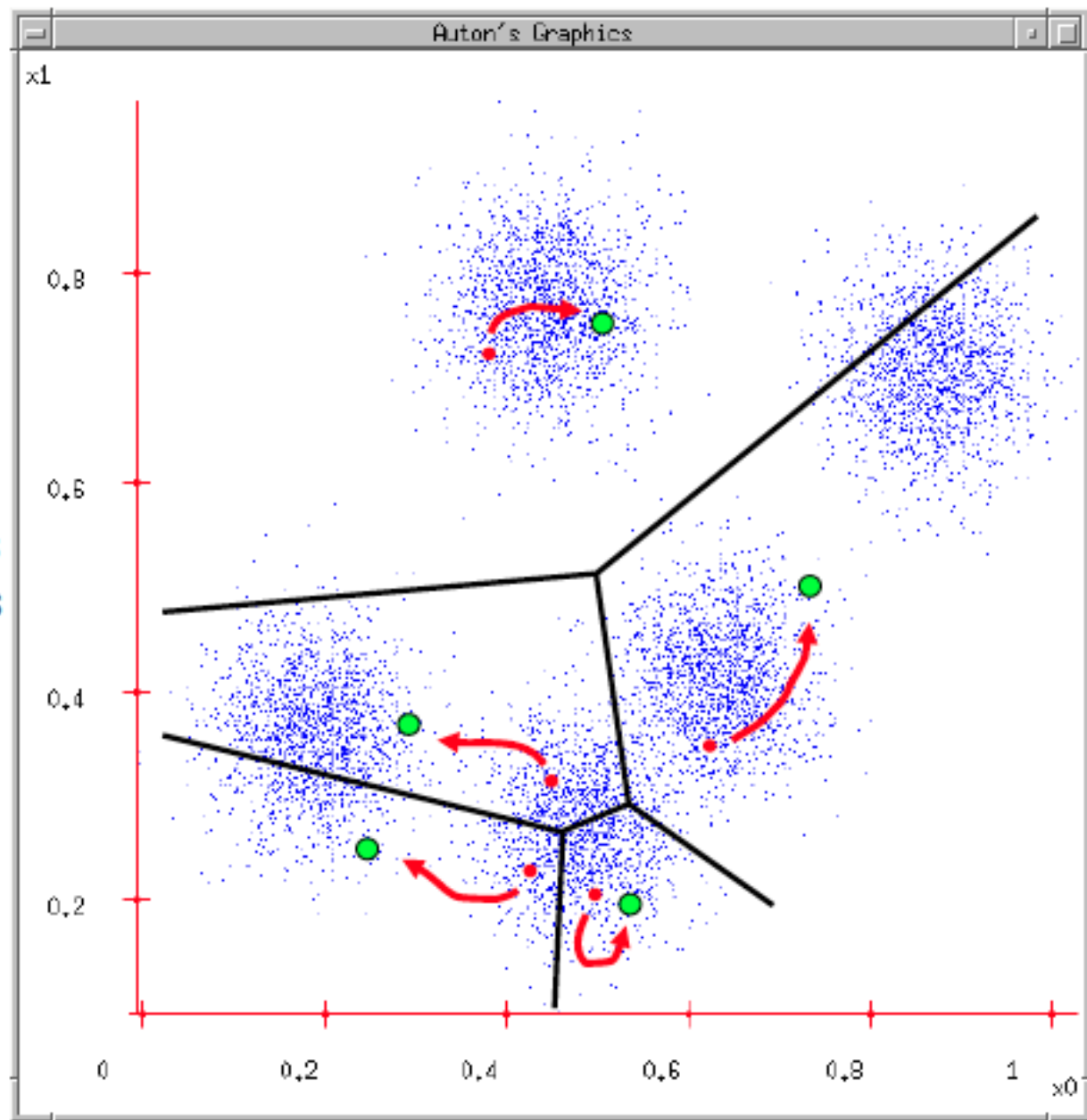
Adapted from Andrew Moore,
<http://www.cs.cmu.edu/~awm/tutorials>

Copyright © 2001, 2004, Andrew W. Moore

K-means and Hierarchical Clustering: Slide 8

K-means

1. Ask user how many clusters they'd like.
(e.g. $k=5$)
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns



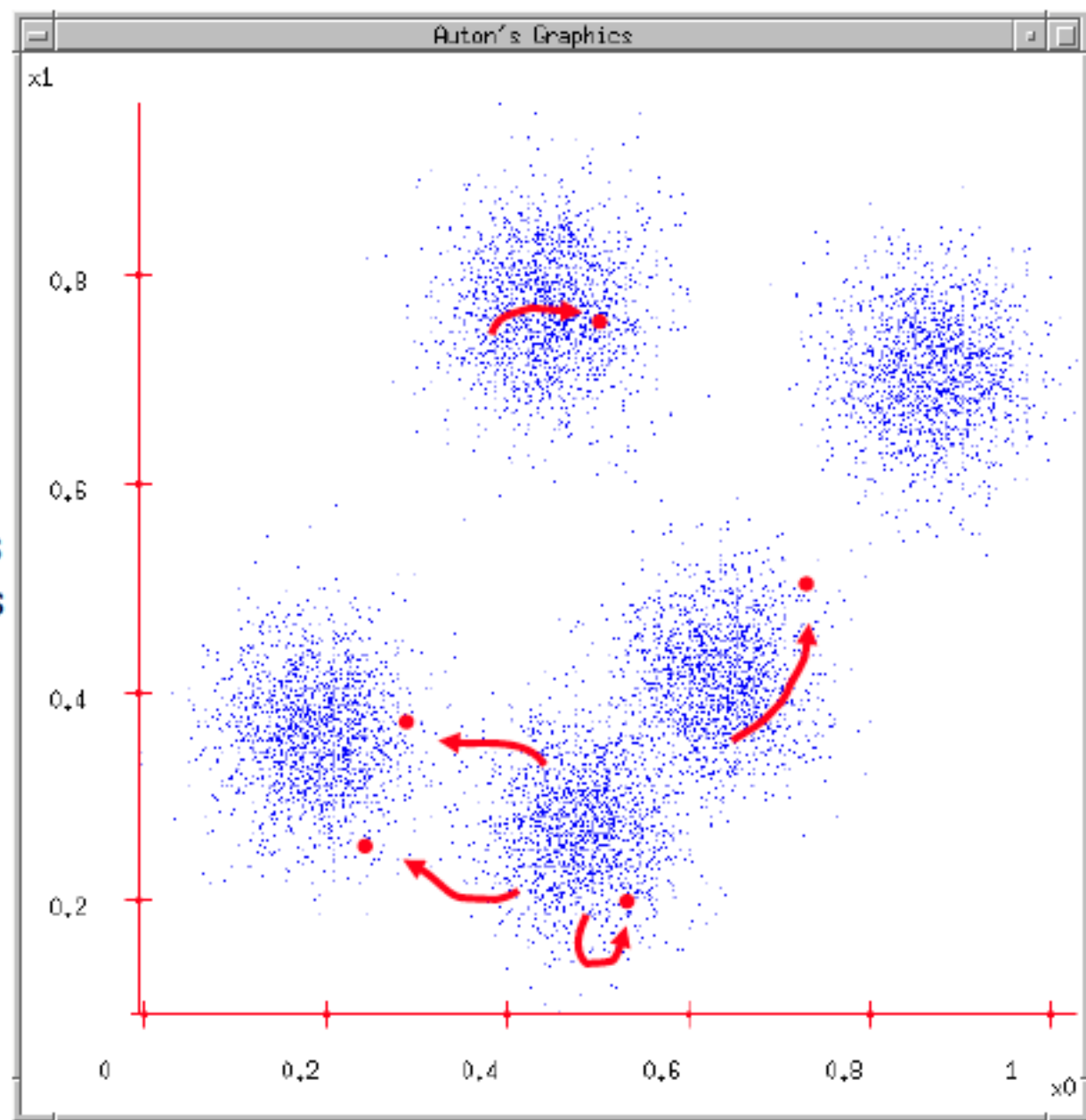
Adapted from Andrew Moore,
<http://www.cs.cmu.edu/~awm/tutorials>

Copyright © 2001, 2004, Andrew W. Moore

K-means and Hierarchical Clustering: Slide 9

K-means

1. Ask user how many clusters they'd like.
(e.g. $k=5$)
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns...
5. ...and jumps there
6. ...Repeat until terminated!



K-means Clustering Algorithm Pseudocode

Algorithm	Basic K-means algorithm.
------------------	--------------------------

- 1: Select K points as initial centroids.
 - 2: **repeat**
 - 3: Form K clusters by assigning each point to its closest centroid.
 - 4: Recompute the centroid of each cluster.
 - 5: **until** Centroids do not change.
-

Distance metric: Chosen by user.

For numerical attributes, often use L_2 (Euclidean) distance: $d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$

Centroid of a cluster here refers to the *mean* of the points in the cluster.

(*) NB: Using a different distance function other than (squared) Euclidean distance may stop the algorithm from converging. Various modifications of k-means such as *spherical k-means* have been proposed to allow using other distance measures.

Example: Image segmentation by K -means clustering by color

From <http://vitroz.com/Documents/Image%20Segmentation.pdf>

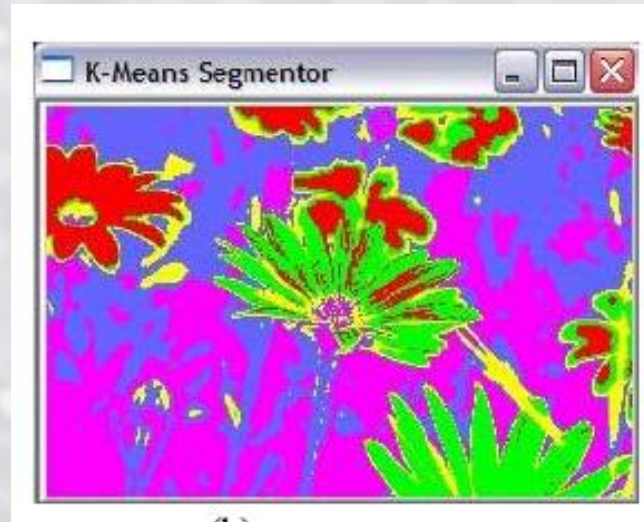
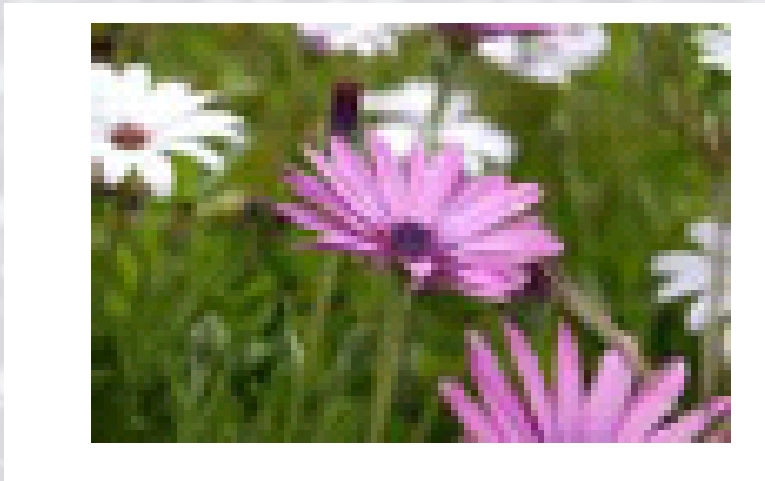
$K=5$, RGB space



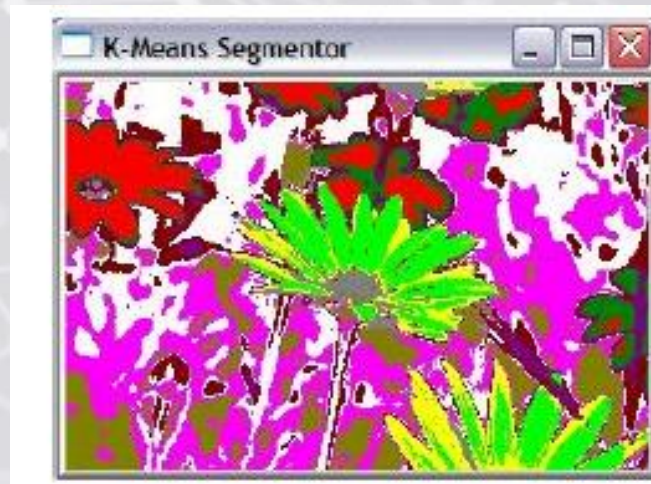
$K=10$, RGB space



$K=5$, RGB space

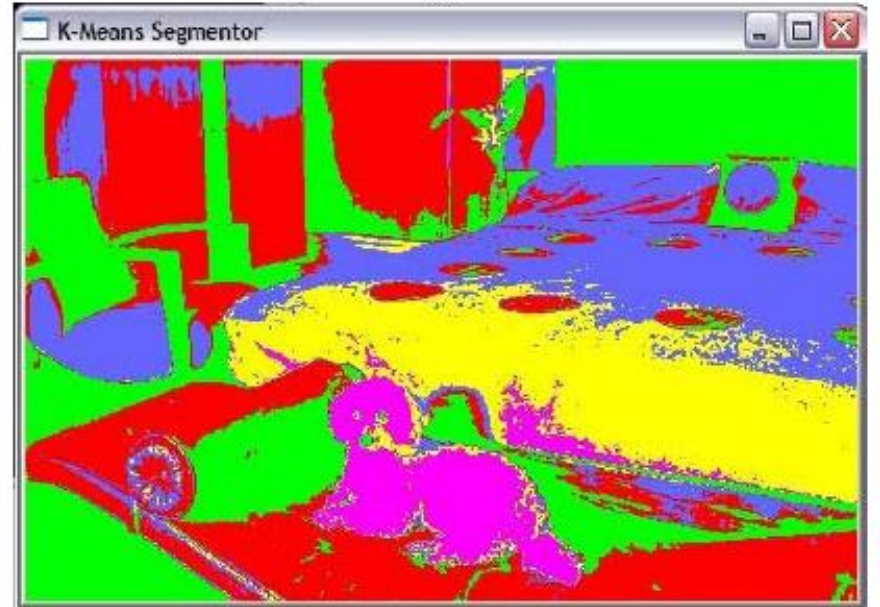


$K=10$, RGB space

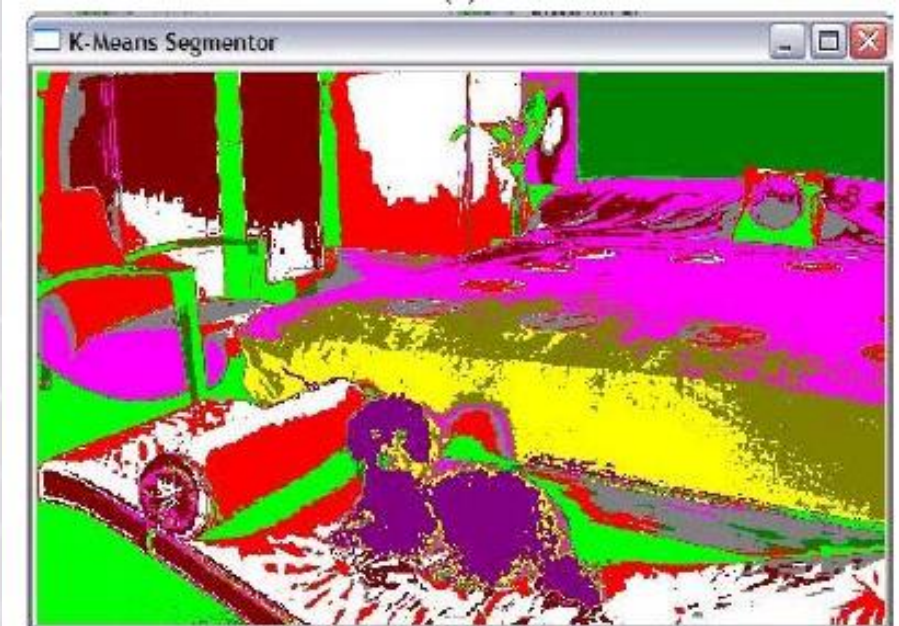


$K=5$, RGB space

(c)



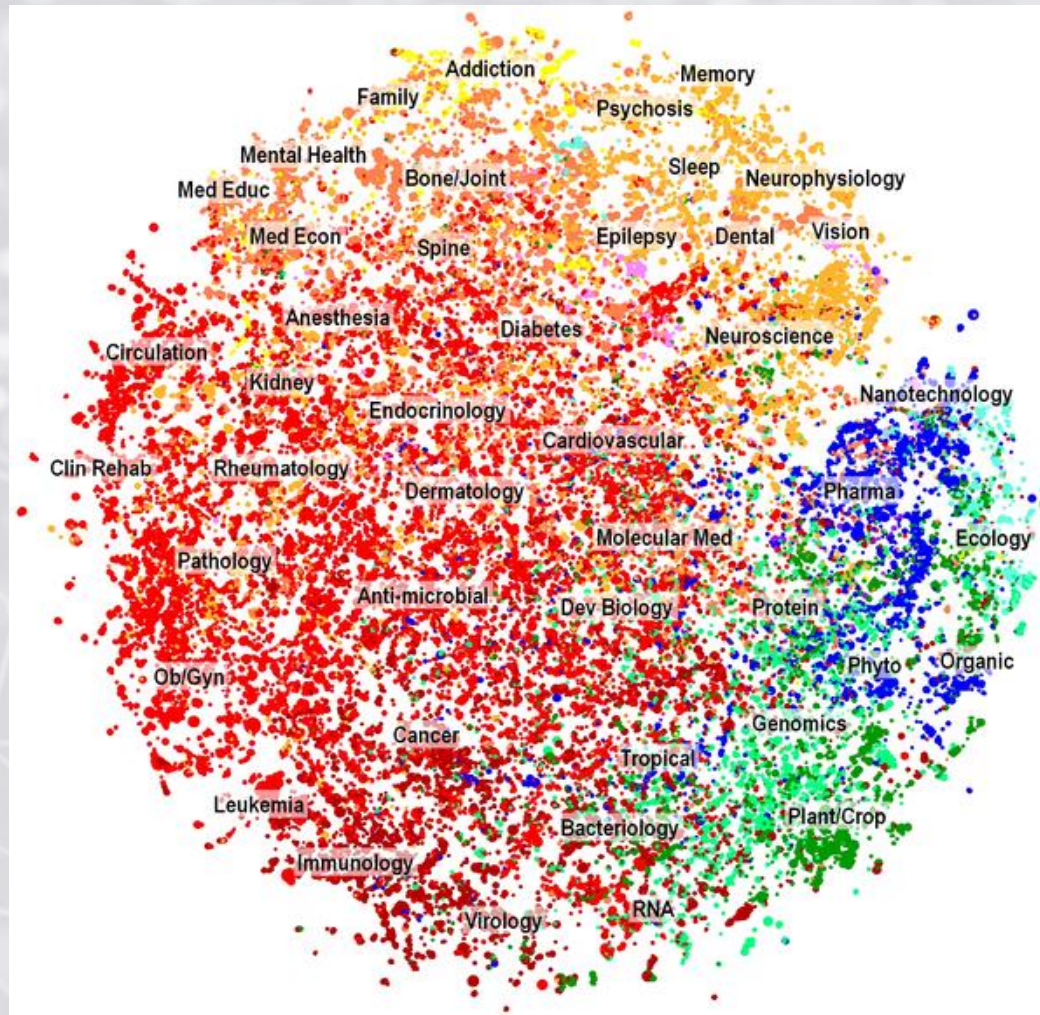
$K=10$, RGB space



Example: Clustering text documents

- A text document is represented as a feature vector of word frequencies (see: *Word2vec*).
- Distance between two documents is the cosine of the angle between their corresponding feature vectors.

Figure 4. Two-dimensional map of the PMRA cluster solution, representing nearly 29,000 clusters and over two million articles.



Boyack KW, Newman D, Duhon RJ, Klavans R, et al. (2011) Clustering More than Two Million Biomedical Publications: Comparing the Accuracies of Nine Text-Based Similarity Approaches. *PLoS ONE* 6(3): e18029. doi:10.1371/journal.pone.0018029
<http://www.plosone.org/article/info:doi/10.1371/journal.pone.0018029>

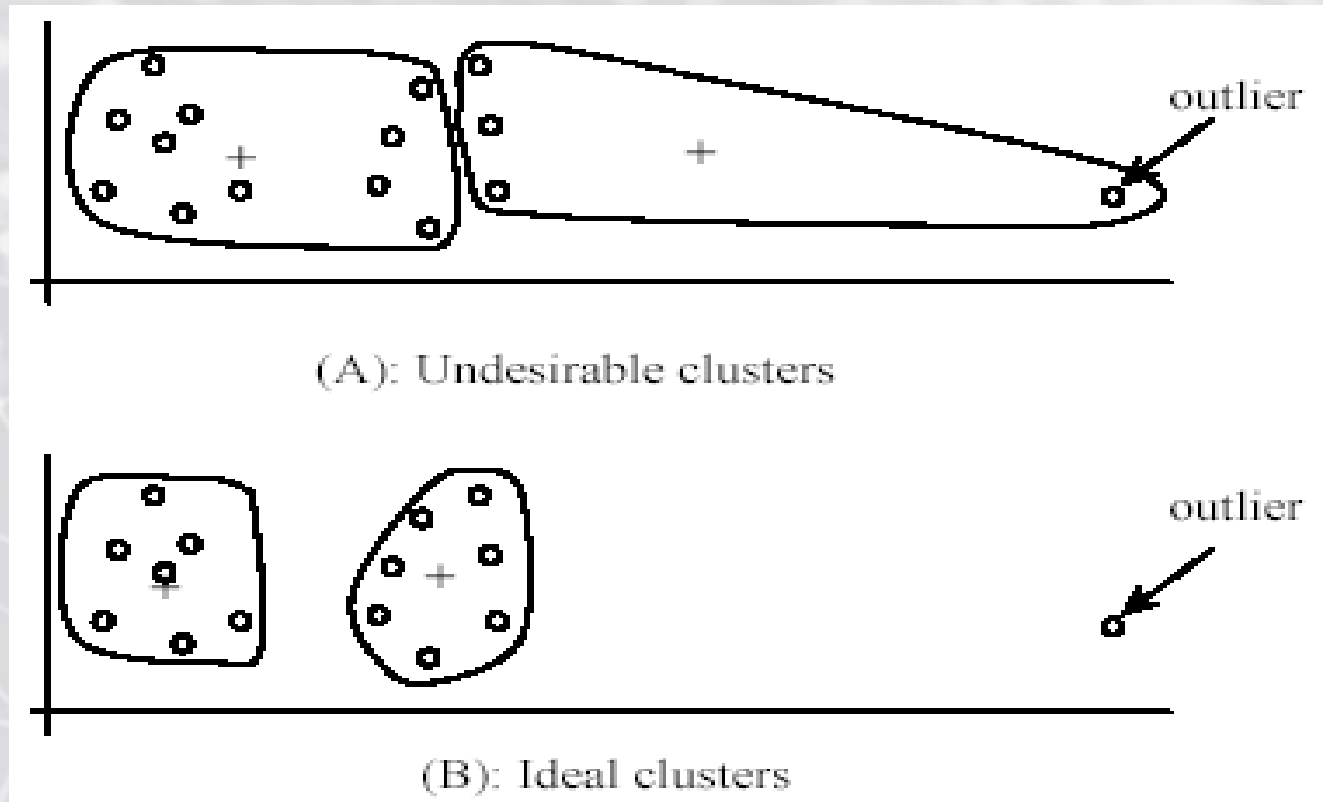
K-means Analysis

- Convergence: k-means is guaranteed to converge in a finite number of steps (irrespective of the initial centroid assignment). Why? In short: there are only a finite number of ways to cluster n data points into k clusters (although note that this number can be large). Usually convergence is relatively fast in practice.
- *NB*: The algorithm is nevertheless not guaranteed to generate a (globally) optimal clustering.
- Complexity: In general, finding the optimal solution to k-means for observations in d dimensions is **NP-hard** (even in the 2-class case).
- The run-time of k-means is $O(nkdi)$, where n is the size of the data set, d is the dimension, k is the number of clusters and i is the number of clusters needed until convergence. k-means is therefore oftentimes considered a linear run-time algorithm; in the worst-case it is more aptly described as superpolynomial.

Potential Issues for K -means

- The algorithm is only applicable if the mean is defined.
 - For categorical data, use **K-modes**: The centroid is represented by the most frequent values.
- The user needs to specify K .
- Cluster morphology can be severely limited (epsilon-balls, etc.)
- Algorithms makes hard cluster assignments (either element belongs to a particular cluster or it does not).
- The algorithm is sensitive to outliers
 - Outliers are data points that are very far away from expected range/other data points.
 - Outliers could be errors in the data recording or some special data points with very different values.
 - Note the *mode* is a more **robust measure of center** (than, say the mean), meaning it is less susceptible to outliers; thus, we can potentially use K-modes to safeguard against influence of outliers.

Issues for K -means: Problems with outliers

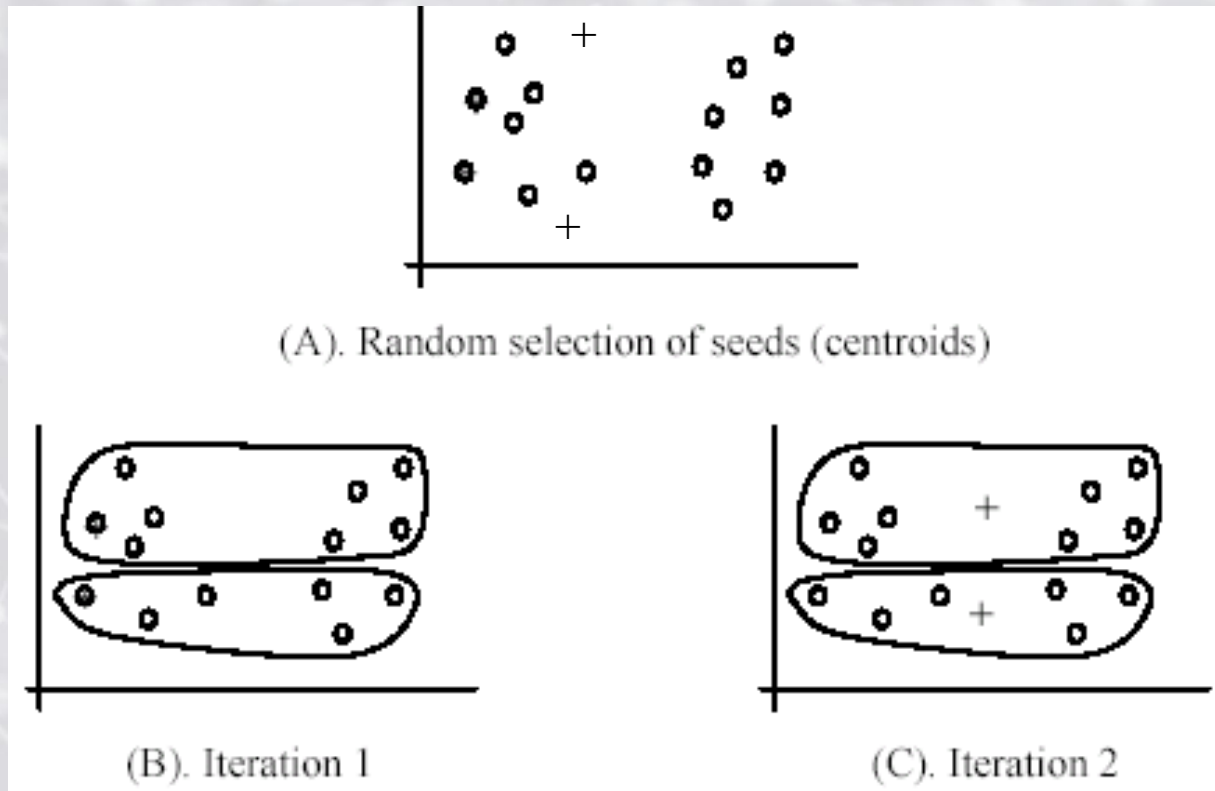


Dealing with outliers

- One method is to remove some data points in the clustering process that are much further away from the centroids than other data points.
 - Expensive
 - Not always a good idea!
- Another method is to perform random sampling. Since in sampling we only choose a small subset of the data points, the chance of selecting an outlier is very small.
 - Assign the rest of the data points to the clusters by distance or similarity comparison, or classification

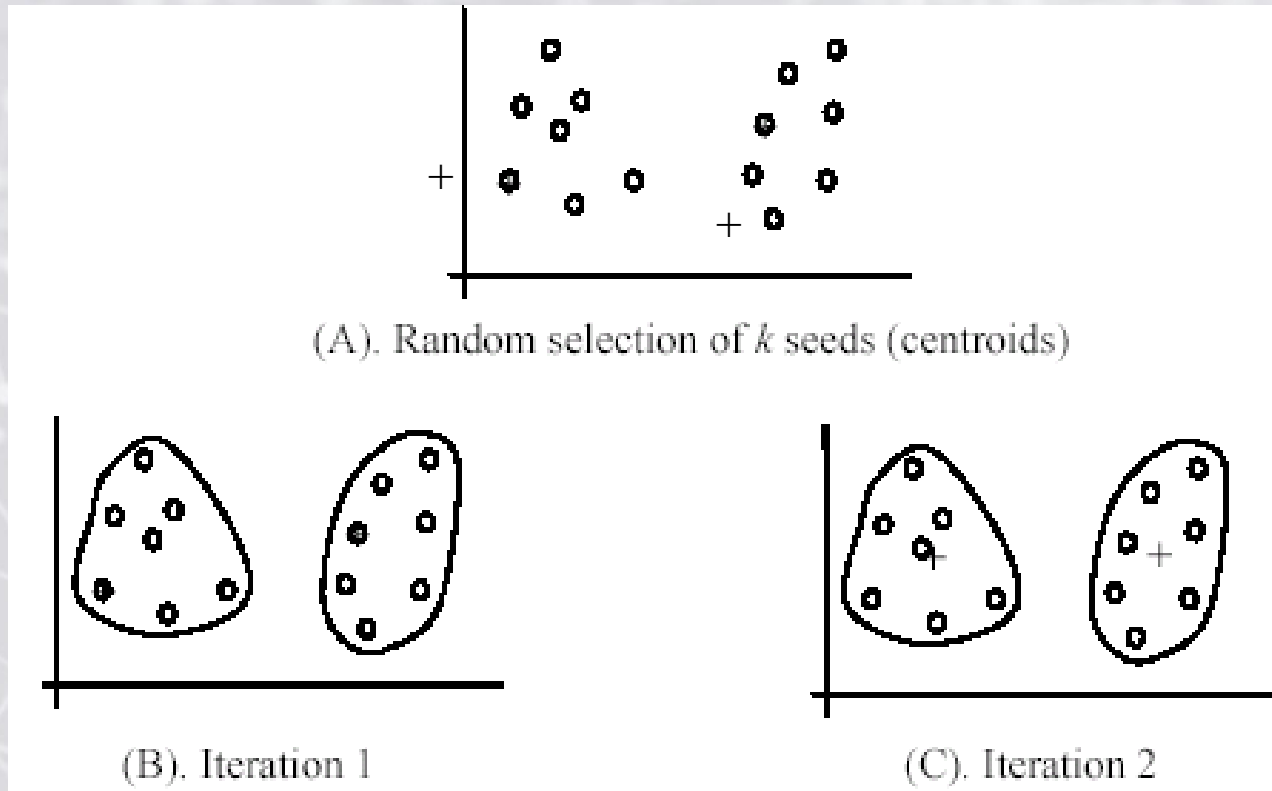
Issues for K -means (cont ...)

- The algorithm is sensitive to **initial seeds**.



Issues for K -means (cont ...)

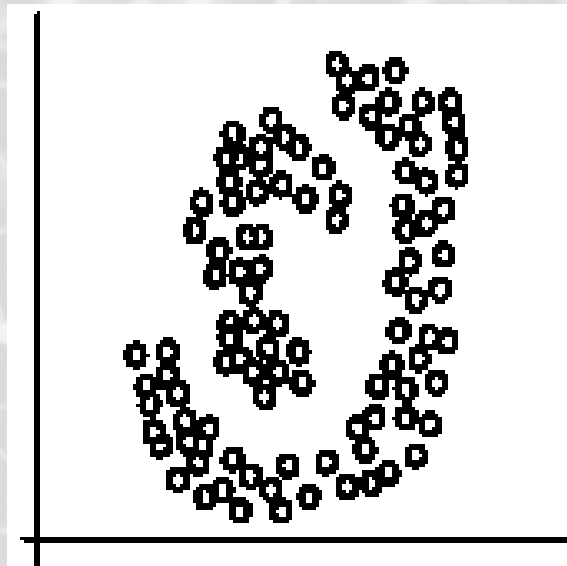
- If we use **different seeds**: good results



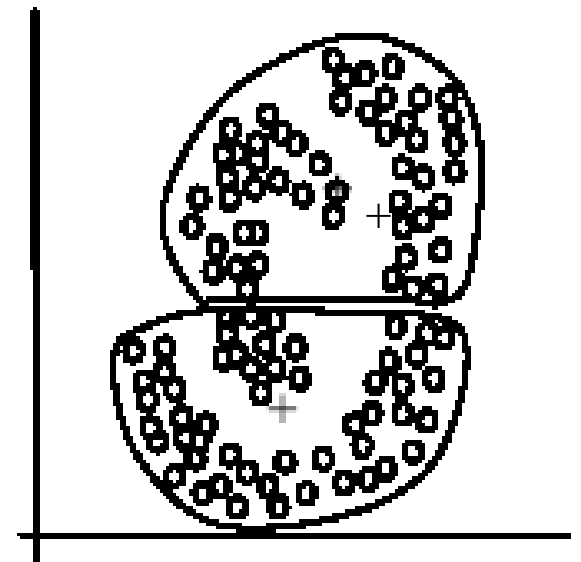
- Often we can improve k -means results by performing **several random restarts**.
- It is commonly helpful to use actual data values for the initial seeds.

Issues for K -means (cont ...)

- The K -means algorithm is not suitable for discovering clusters that are not hyper-ellipsoids (or hyper-spheres).



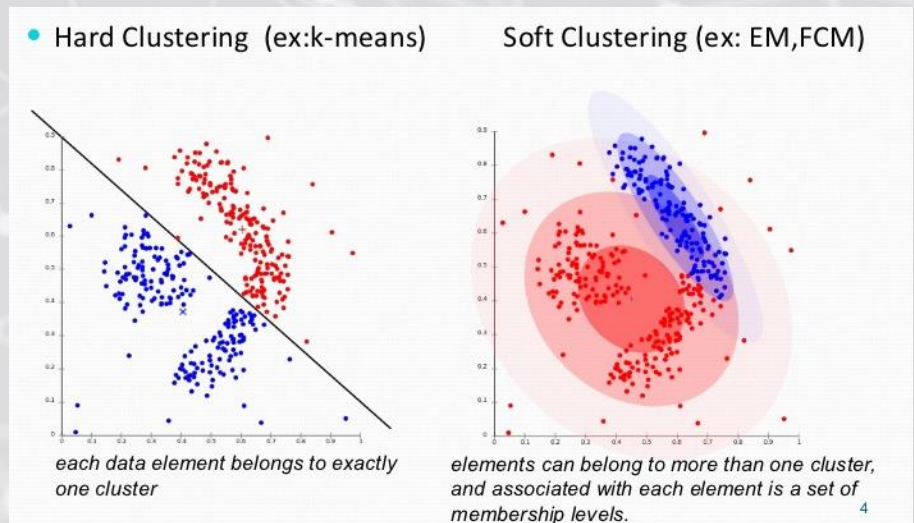
(A): Two natural clusters



(B): k -means clusters

Fuzzy c-means

- In non-fuzzy clustering (also known as hard clustering), data is divided into distinct clusters, where each data point can only belong to exactly one cluster. (cf., k-means from previous slides).
- In *fuzzy clustering* (also: soft clustering), data points can potentially belong to multiple clusters.
- Commonly, “**membership grades**” (i.e. class probabilities) are assigned to each of the data points. These membership grades indicate the degree to which data points belong to each cluster. Thus, points on the edge of a cluster, with lower membership grades, may be *in the cluster* to a lesser degree than points in the center of cluster.



Fuzzy c-means

- The **FCM** (*fuzzy c-means* (1973), as it is usually called) algorithm is very similar to the k-means algorithm:


Here is the basic idea:

- Choose a number of clusters: c (a hyperparameter).
- Initially assign coefficients randomly to each data point for being in the clusters (these are the *initial membership grades*).
- Repeat until the algorithm has converged/stopping condition:
 - (I) Compute the centroid for each cluster (m-step).
 - (II) For each data point, compute its coefficients/membership grades for being in the clusters (e-step).

Fuzzy c-means

Here is the basic idea:

- Choose a number of clusters: c (a hyperparameter).
- Initially assign coefficients randomly to each data point for being in the clusters (these are the *initial membership grades*).
- Repeat until the algorithm has converged/stopping condition:
 - (I) Compute the centroid for each cluster (m-step).
 - (II) For each data point, compute its coefficients/membership grades for being in the clusters (e-step).



$$\mathbf{c}_k = \frac{\sum_x w_k(\mathbf{x})^m \mathbf{x}}{\sum_x w_k(\mathbf{x})^m}$$


where $\mathbf{C} = \{\mathbf{c}_1, \dots, \mathbf{c}_c\}$ are the cluster centers, $W = w_{i,j} \in [0,1], i=1, \dots, n, j=1, \dots, c$, and each element w_{ij} tells the degree to which element \mathbf{x}_i belongs to cluster \mathbf{c}_j (i.e. the w 's are the *membership grades*); $m > 1$ is a hyperparameter known as the **fuzzifier parameter** which controls the amount of “fuzziness” in the partition.

Fuzzy c-means

Here is the basic idea:

- Choose a number of clusters: c (a hyperparameter).
- Initially assign coefficients randomly to each data point for being in the clusters (these are the *initial membership grades*).
- Repeat until the algorithm has converged/stopping condition:
 - (I) Compute the centroid for each cluster (m-step).
 - (II) For each data point, compute its coefficients/membership grades for being in the clusters (e-step).



$$\mathbf{c}_k = \frac{\sum_x w_k (\mathbf{x})^m \mathbf{x}}{\sum_x w_k (\mathbf{x})^m}$$



$$w_{ij} = \frac{1}{\sum_{k=1}^c \left(\frac{\|\mathbf{x}_i - \mathbf{c}_j\|}{\|\mathbf{x}_i - \mathbf{c}_k\|} \right)^{\frac{2}{m-1}}}$$

Fuzzy c-means

Here is the basic idea:

- Choose a number of clusters: c (a hyperparameter).
- Initially assign coefficients randomly to each data point for being in the clusters (these are the *initial membership grades*).
- Repeat until the algorithm has converged/stopping condition:
 - (I) Compute the centroid for each cluster (m-step).
 - (II) For each data point, compute its coefficients/membership grades for being in the clusters (e-step).


$$\mathbf{c}_k = \frac{\sum_x w_k (\mathbf{x})^m \mathbf{x}}{\sum_x w_k (\mathbf{x})^m}$$


$$w_{ij} = \frac{1}{\sum_{k=1}^c \left(\frac{\|\mathbf{x}_i - \mathbf{c}_j\|}{\|\mathbf{x}_i - \mathbf{c}_k\|} \right)^{\frac{2}{m-1}}}$$

(*) FCM aims to minimize the an objective function:

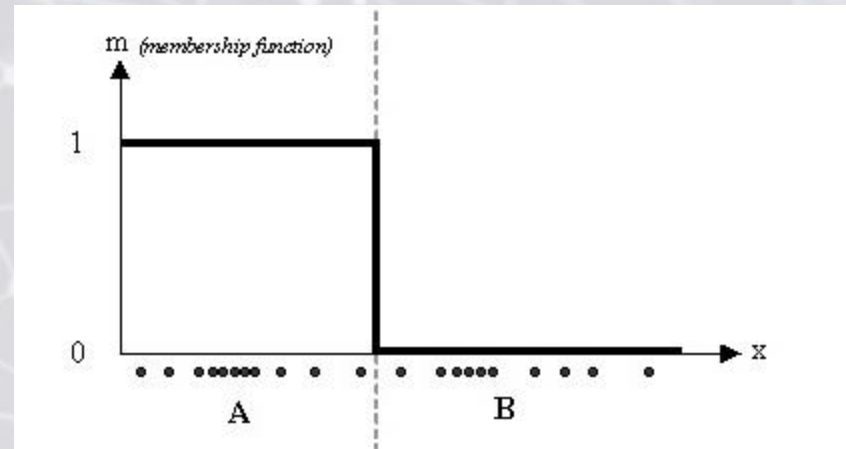
$$\arg \min_C \sum_{i=1}^n \sum_{j=1}^c w_{ij}^m \|\mathbf{x}_i - \mathbf{c}_j\|^2$$

Fuzzy c-means

- For example, consider a simple 1-d data set, where we want to determine a plausible clustering for 2 classes (A and B).



- Using “**hard k-means**” we associate each datum to a specific centroid; therefore the membership function looks like this:

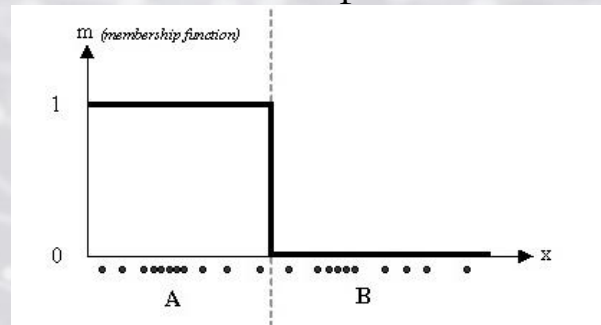


Fuzzy c-means

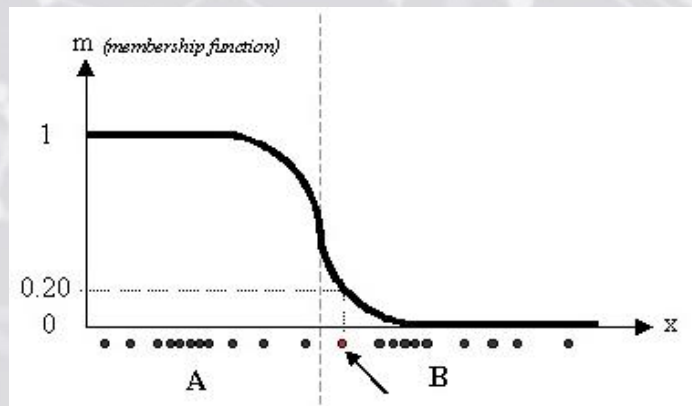
- For example, consider a simple 1-d data set, where we want to determine a plausible clustering for 2 classes (A and B).



- Using “**hard k-means**” we associate each datum to a specific centroid; therefore the membership function looks like this:

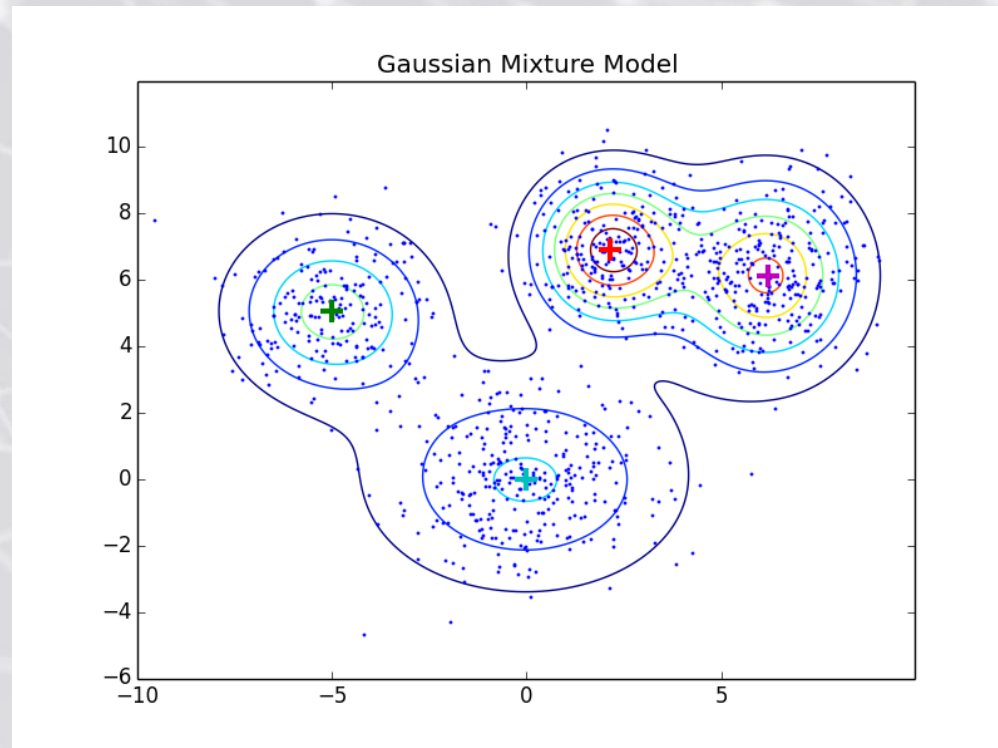


- In the **FCM approach**, instead, the same given datum does not belong exclusively to a well defined cluster. In this case, the membership function follows a smoother line to indicate that every datum may belong to several clusters with different values of the membership coefficient.



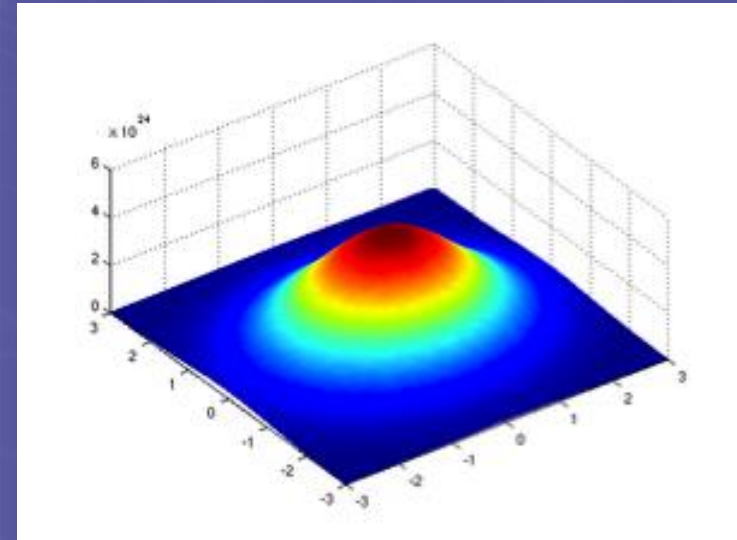
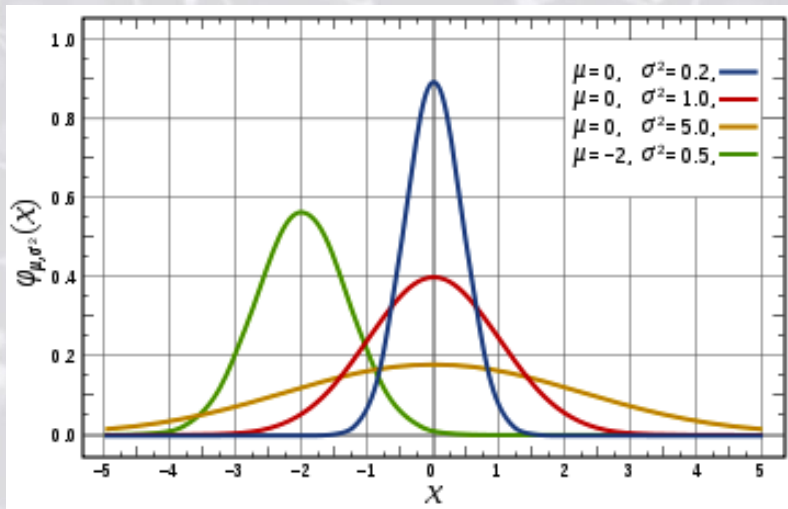
GMMs

- FCM was essentially, again, a “fuzzy” version of the k-means algorithm, where data points are assigned to each cluster with an associated probability/membership grade.
- An additional, commonly used soft clustering model is the GMM (**Gaussian mixture model**); with GMMs, we assume (*a priori*) that the clusters resemble tightly-packed balls (i.e. Gaussian distributions).

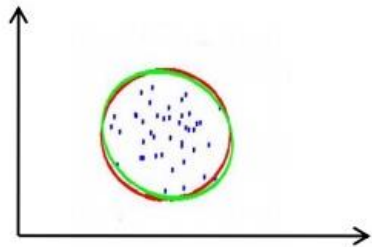


GMMs: Gaussian Distribution Review

$$N(x; \mu, \Sigma) = \frac{1}{(2\pi)^{D/2} |\Sigma|^{1/2}} \exp \left\{ -\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right\}$$

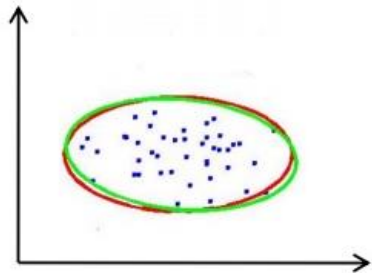


GMMs: Gaussian Distribution Review



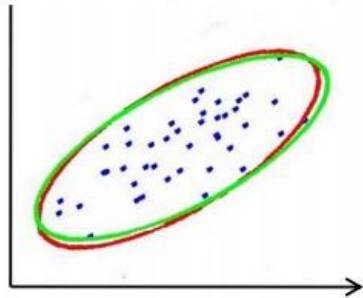
Covariance matrix $\Sigma =$

$$\begin{matrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{matrix}$$



Covariance matrix $\Sigma =$

$$\begin{matrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{matrix}$$



Covariance matrix $\Sigma =$

$$\begin{matrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{matrix}$$

Using different forms of covariance matrix allows for clusters of different shapes

GMMs

Main ideas for clustering using GMM:

(*) *Initialization*: given a data set, fix k , the number of clusters; initialize the mean (μ) and covariance matrices (Σ) for the k Gaussian clusters.

(*) Assign the data points to the k clusters (using a soft clustering) (**assignment step/E-step**)

(*) Update the parameters (i.e. μ , Σ) for each of the clusters. (**update step/M-step**)

...repeat until stopping condition/convergence

GMMs

Main ideas for clustering using GMM:

- (*) *Initialization*: given a data set, fix k , the number of clusters; initialize the mean (μ) and covariance matrices (Σ) for the k Gaussian clusters.
 - (*) Assign the data points to the k clusters (using a soft clustering) (**assignment step/E-step**)
 - (*) Update the parameters (i.e. μ , Σ) and prior class estimates ($P(C_i | x)$) (for each of the clusters). (**update step/M-step**)
- ...repeat until stopping condition/convergence

What makes this problem challenging? There are, ostensibly, **many unknowns!**

- (*) Strictly speaking, we don't know the cluster assignments nor any of the Gaussian distribution parameters.

GMMs

What makes this problem challenging? There are, ostensibly, **many unknowns!**

(*) Strictly speaking, we don't know the cluster assignments nor any of the Gaussian distribution parameters.

How can we simplify things?

A nice trick...Solve each subproblem separately!

- (1) For instance, to find the optimal class assignments for each datum, use the current approximations for the Gaussian parameters distributions (i.e. treat μ and Σ as known for each cluster, as well as each class prior) and compute the class posterior: $P(C_i | x)$ using Bayes' Rule.
- (2) Conversely, to find the optimal estimates for μ and Σ for each cluster, in addition to the class priors, use the current (soft) class posterior assignments and compute the MLE.

GMMs

(1) For instance, to find the optimal class assignments for each datum, use the current approximations for the Gaussian parameters distributions (i.e. treat μ and Σ as known for each cluster, as well as each class prior) and compute the class posterior: $P(C_i | x)$ using Bayes' Rule. (**assignment step/E-step**)

(*) Given the current estimates of both the parameters of each Gaussian cluster: $(\mu_1, \Sigma_1), \dots, (\mu_k, \Sigma_k)$, and the prior for each cluster: $P(C_1) = \pi_1, \dots, P(C_k) = \pi_k$, we compute the class posterior $P(C_i)$ using Bayes' Rule as follows:

$$P(C_i | x) = \frac{P(x | C_i) P(C_i)}{P(x)}$$

GMMs

- (1) For instance, to find the optimal class assignments for each datum, use the current approximations for the Gaussian parameters distributions (i.e. treat μ and Σ as known for each cluster, as well as each class prior) and compute the class posterior: $P(C_i | x)$ using Bayes' Rule. (**assignment step/E-step**)

(*) Given the current estimates of both the parameters of each Gaussian cluster: $(\mu_1, \Sigma_1), \dots, (\mu_k, \Sigma_k)$, and the prior for each cluster: $P(C_1) = \pi_1, \dots, P(C_k) = \pi_k$, we compute the class posterior $P(C_i)$ using Bayes' Rule as follows:

$$P(C_i | x) = \frac{P(x | C_i) P(C_i)}{P(x)} \propto \pi_i \frac{1}{(2\pi)^{d/2} |\Sigma_i|^{1/2}} \exp \left[-\frac{1}{2} (x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i) \right]$$

GMMs

(2) To find the optimal estimates for μ and Σ for each cluster, in addition to the class priors, use the current (soft) class posterior assignments and compute the MLE. (**update step/M-step**)

(*) Observe that if we knew which points belong to, say cluster i , for a hard clustering, we can use the standard MLE estimates (from beginning statistics) to estimate the Gaussian parameters (μ and Σ) for each cluster, in addition to the cluster priors (e.g. $P(C_i)$). These standard parameter estimates are given as follows:

$$\begin{array}{ccc} \hat{\pi}_i = \frac{n_i}{n} & \hat{\mu}_i = \frac{1}{n_i} \sum_{\mathbf{x}^j \in C_i} \mathbf{x}^j & \hat{\Sigma}_i = \frac{1}{n_i} \sum_{\mathbf{x}^j \in C_i} (\mathbf{x}^j - \hat{\mu}_i)(\mathbf{x}^j - \hat{\mu}_i)^T \\ \text{cluster prior} & \text{cluster mean} & \text{cluster covariance matrix} \end{array}$$

where above, n_i denotes the size of the i th cluster.

GMMs: MLE Parameter Estimates

(2) To find the optimal estimates for μ and Σ for each cluster, in addition to the class priors, use the current (soft) class posterior assignments and compute the MLE. (**update step/M-step**)

(*) Observe that if we knew which points belong to, say cluster i , for a hard clustering, we can use the standard MLE estimates (from beginning statistics) to estimate the Gaussian parameters (μ and Σ) for each cluster, in addition to the cluster priors (e.g. $P(C_i)$). These standard parameter estimates are given as follows:

$$\begin{array}{ccc} \hat{\pi}_i = \frac{n_i}{n} & \hat{\mu}_i = \frac{1}{n_i} \sum_{\mathbf{x}^j \in C_i} \mathbf{x}^j & \hat{\Sigma}_i = \frac{1}{n_i} \sum_{\mathbf{x}^j \in C_i} (\mathbf{x}^j - \hat{\mu}_i)(\mathbf{x}^j - \hat{\mu}_i)^T \\ \text{cluster prior} & \text{cluster mean} & \text{cluster covariance matrix} \end{array}$$

where above, n_i denotes the size of the i th cluster.

(*) However, because we are executing a soft clustering, these parameter update formulae must incorporate the class posteriors: $P(C_i | \mathbf{x})$, for each $i=1, \dots, k$ and for each data point \mathbf{x} , respectively.

GMMs: Modified Parameter Estimates

(2) To find the optimal estimates for μ and Σ for each cluster, in addition to the class priors, use the current (soft) class posterior assignments and compute the MLE. (**update step/M-step**)

(*) Here are the parameter estimate formulas, updated to account for the soft clustering induced by the class posteriors: $P(C_i | \mathbf{x})$, for each $i=1, \dots, k$, for each data point:

$$\hat{\pi}_i = \frac{n_i}{n} \rightarrow \hat{\pi}_i = \frac{1}{n} \sum_{j=1, \dots, n} P(C_i | \mathbf{x}^j)$$

cluster prior modified formula

$$\hat{\mu}_i = \frac{1}{n_i} \sum_{\mathbf{x}^j \in C_i} \mathbf{x}^j \rightarrow \hat{\mu}_i = \frac{\sum_{j=1, \dots, n} \mathbf{x}^j P(C_i | \mathbf{x}^j)}{\sum_{j=1, \dots, n} P(C_i | \mathbf{x}^j)}$$

cluster mean modified formula

$$\hat{\Sigma}_i = \frac{1}{n_i} \sum_{\mathbf{x}^j \in C_i} (\mathbf{x}^j - \hat{\mu}_i)(\mathbf{x}^j - \hat{\mu}_i)^T \rightarrow \hat{\Sigma}_i = \frac{\sum_{j=1, \dots, n} P(C_i | \mathbf{x}^j) (\mathbf{x}^j - \hat{\mu}_i)(\mathbf{x}^j - \hat{\mu}_i)^T}{\sum_{j=1, \dots, n} P(C_i | \mathbf{x}^j)}$$

cluster covariance matrix modified formula

GMMs: Summary

Main ideas for clustering using GMM:

(*) *Initialization*: given a data set, fix k , the number of clusters; initialize the mean (μ) and covariance matrices (Σ) for the k Gaussian clusters, and cluster priors ($P(C_i)$).

(I) Assign the data points to the k clusters (using a soft clustering) (**assignment step/E-step**)

$$P(C_i | x) \propto \pi_i \frac{1}{(2\pi)^{d/2} |\Sigma_i|^{1/2}} \exp \left[-\frac{1}{2} (x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i) \right]$$

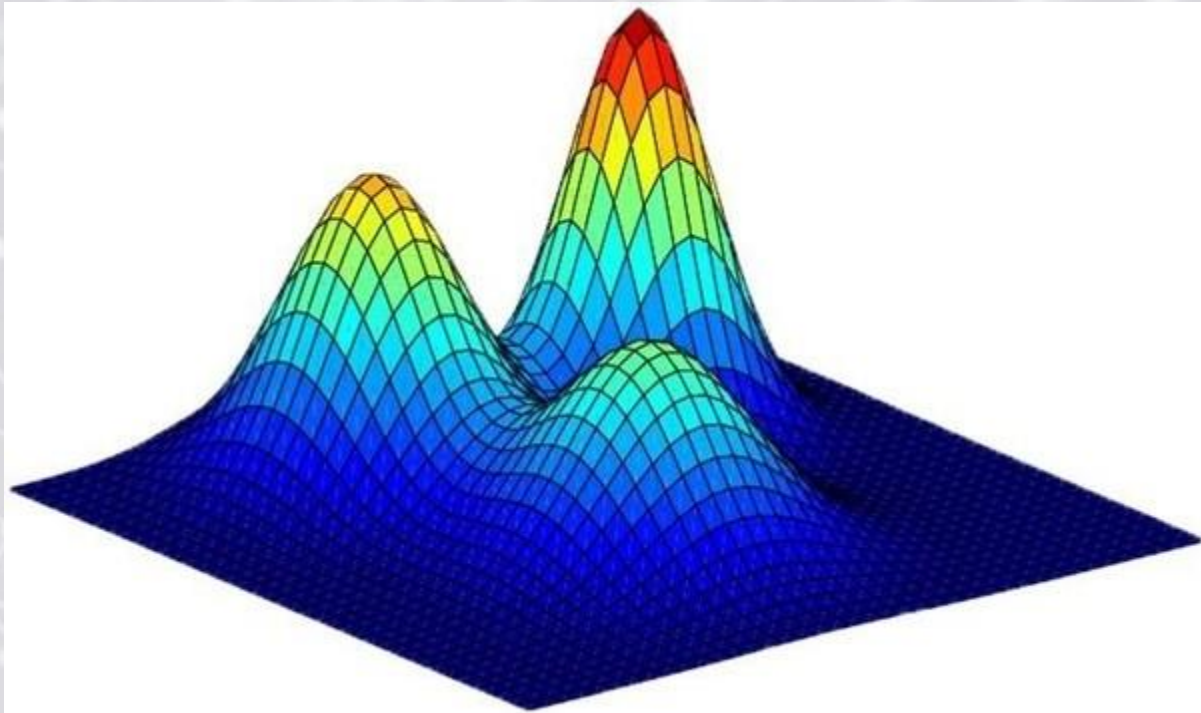
(II) Update the parameters (i.e. μ , Σ) for each of the clusters, including the cluster priors. (**update step/M-step**)

$$\hat{\pi}_i = \frac{1}{n} \sum_{j=1, \dots, n} P(C_i | \mathbf{x}^j) \quad \hat{\mu}_i = \frac{\sum_{j=1, \dots, n} \mathbf{x}^j P(C_i | \mathbf{x}^j)}{\sum_{j=1, \dots, n} P(C_i | \mathbf{x}^j)} \quad \hat{\Sigma}_i = \frac{\sum_{j=1, \dots, n} P(C_i | \mathbf{x}^j) (\mathbf{x}^j - \hat{\mu}_i)(\mathbf{x}^j - \hat{\mu}_i)^T}{\sum_{j=1, \dots, n} P(C_i | \mathbf{x}^j)}$$

...repeat until stopping condition/convergence

GMMs

- Demo: <https://lukapopijac.github.io/gaussian-mixture-model/>



Cluster Analysis: Measuring Cluster Validity

- Previously we considered clustering in a commonplace, unsupervised setting (i.e. no class labels); it is also likewise useful to consider clustering for supervised learning.

When labels are present, we can determine a *post hoc* evaluation of **cluster validity** (namely, the “goodness” of the clustering model).

Why might we want to do this?

Cluster Analysis: Measuring Cluster Validity

- Previously we considered clustering in a commonplace, unsupervised setting (i.e. no class labels); it is also likewise useful to consider clustering for supervised learning.

When labels are present, we can determine a *post hoc* evaluation of **cluster validity** (namely, the “goodness” of the clustering model).

Why might we want to do this? There are a number of reasons, including:

- (*) Determining whether non-random structure actually exists in the data set
- (*) Comparing cluster results to externally known results (the labels, for instance)
- (*) Comparing the results of two different clustering algorithms
- (*) Determining a good value of the number of clusters (e.g. k)

Cluster Analysis: Measuring Cluster Validity

- There are a great many different numerical measures that can be used to quantify and assess cluster validity; in general these measures fall into at least (2) basic categories:

(*) **Internal index**: Used to measure quality of clustering without reference to external information, e.g., *MSE*, *MSS*.

(*) **External index**: Used to measure degree to which cluster labels match class labels, e.g., *purity*, *entropy*.

Measuring Cluster Validity: Internal Indices (MSE)

(*) In an unsupervised setting we can still measure and assess cluster validity by using internal indices.

For example, let C denote a clustering (i.e. a set of k clusters resulting from a clustering algorithm) and let c denote a particular cluster in C , and define $|c|$ as the number of elements in that cluster.

- We want to minimize the distance between elements of c and the centroid μ_c – this would give us a naturally plausible clustering; formally we minimize **mean square error (MSE)**:

$$MSE(c) = \frac{\sum_{\mathbf{x} \in c} d(\mathbf{x}, \mu_c)^2}{|c|}$$

$$Average\ MSE(C) = \frac{\sum_{c \in C} mse(c)}{K}$$

Measuring Cluster Validity: Internal Indices (MSS)

(*) In an unsupervised setting we can still measure and assess cluster validity by using internal indices.

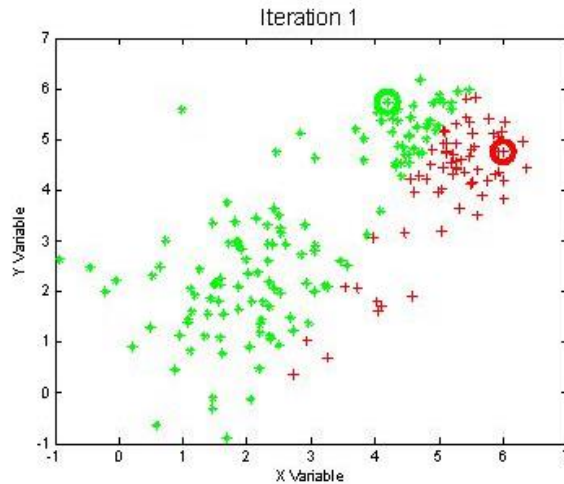
For example, let C denote a clustering (i.e. a set of k clusters resulting from a clustering algorithm) and let c denote a particular cluster in C , and define $|c|$ as the number of elements in that cluster.

• On the other hand, we also want to maximize the pairwise separation of each cluster. That is, **maximize mean square separation (MSS)**:

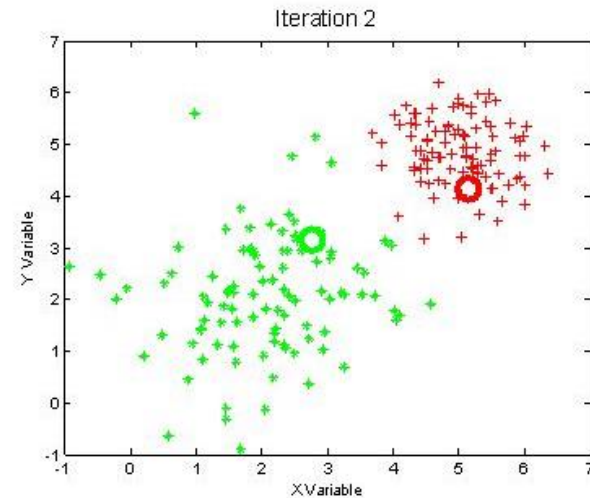
$$MSS(C) = \frac{\sum_{\text{all distinct pairs of clusters } i, j \in C \ (i \neq j)} d(\mu_i, \mu_j)^2}{K(K-1)/2}$$

Measuring Cluster Validity: Internal Indices

Example



MSE Cluster 1 = 1.31
MSE Cluster 2 = 3.21
Overall MSE = 2.57



MSE Cluster 1 = 1.01
MSE Cluster 2 = 1.76
Overall MSE = 1.38

$$MSE(c) = \frac{\sum_{\mathbf{x} \in c} d(\mathbf{x}, \boldsymbol{\mu}_c)^2}{|c|}$$

$$Average\ MSE(C) = \frac{\sum_{c \in C} mse(c)}{K}$$

Measuring Cluster Validity: External Indices

(Entropy)

(*) In an a supervised setting we measure and assess cluster validity by using external indices.

For example, let C denote a clustering (i.e. a set of k clusters resulting from a clustering algorithm) and let c denote a particular cluster in C , and define $|c|$ as the number of elements in that cluster.

- **Entropy of a cluster**: The degree to which a cluster consists of objects of a single class.

$$Entropy(c_i) = - \sum_{j=1}^{|c_i|} p_{i,j} \log_2 p_{i,j}$$

where $p_{i,j}$ denotes the probability that a member of cluster i belongs to cluster j (i.e. $p_{i,j} = n_{i,j} / n_i$, where $n_{i,j}$ represents the number of instances in cluster i with class j , and n_i is the number of instances in cluster i).

$$mean\ Entropy(C) = \sum_{i=1}^K \frac{n_i}{n} Entropy(c_i)$$

*NB: A “good” clustering will have a small mean entropy value.

Measuring Cluster Validity: External Indices

(Entropy)

- **Entropy of a cluster**: The degree to which a cluster consists of objects of a single class.

$$Entropy(c_i) = - \sum_{j=1}^{|c_i|} p_{i,j} \log_2 p_{i,j} \quad \text{mean Entropy}(C) = \sum_{i=1}^K \frac{n_i}{n} Entropy(c_i)$$

where $p_{i,j}$ denotes the probability that a member of cluster i belongs to class j (i.e. $p_{i,j} = n_{i,j} / n_i$, where $n_{i,j}$ represents the number of instances in cluster i with class j , and n_i is the number of instances in cluster i).

Entropy example:

Suppose there are 3 classes: 1, 2, 3

Cluster 1

1 2 1 3 1 1 3

Cluster 2

2 3 3 3 2 3

Cluster 3

1 1 3 2 2 3 2

$$entropy(c_1) = - \left(\frac{4}{7} \log_2 \frac{4}{7} + \frac{1}{7} \log_2 \frac{1}{7} + \frac{2}{7} \log_2 \frac{2}{7} \right) = 1.37$$

$$entropy(c_2) = - \left(0 + \frac{2}{6} \log_2 \frac{2}{6} + \frac{4}{6} \log_2 \frac{4}{6} \right) = 0.91$$

$$entropy(c_3) = - \left(\frac{2}{7} \log_2 \frac{2}{7} + \frac{3}{7} \log_2 \frac{3}{7} + \frac{2}{7} \log_2 \frac{2}{7} \right) = 1.54$$

$$\text{mean entropy}(C) = \frac{7}{20} (1.37) + \frac{6}{20} (0.91) + \frac{7}{20} (1.54)$$

Measuring Cluster Validity: External Indices

(Purity)

- **Purity of a cluster**: Purity is defined as the percent of the total number of data points that were classified correctly, in the unit range $[0,1]$:

$$Purity(C) = \frac{1}{n} \sum_{i=1}^k \max_j |c_i \cap t_j|$$

Where n is the number of data points, k is the number of clusters, c_i is a cluster in C , and t_j is the classification which has the max count for cluster c_i .

(*) To compute purity, each cluster is assigned to the class which is most frequent in the cluster, and then the accuracy of this assignment is measured by counting the number of correctly assigned data and dividing by n .

Measuring Cluster Validity: External Indices

(Purity)

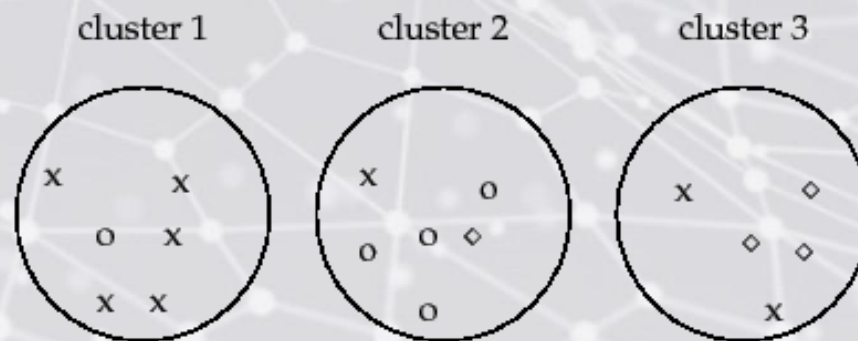
- **Purity of a cluster**: Purity is defined as the percent of the total number of data points that were classified correctly, in the unit range $[0,1]$:

$$Purity(C) = \frac{1}{n} \sum_{i=1}^k \max_j |c_i \cap t_j|$$

Where n is the number of data points, k is the number of clusters, c_i is a cluster in C , and t_j is the classification which has the max count for cluster c_i .

(*) To compute purity, each cluster is assigned to the class which is most frequent in the cluster, and then the accuracy of this assignment is measured by counting the number of correctly assigned data and dividing by n .

Consider the following example:

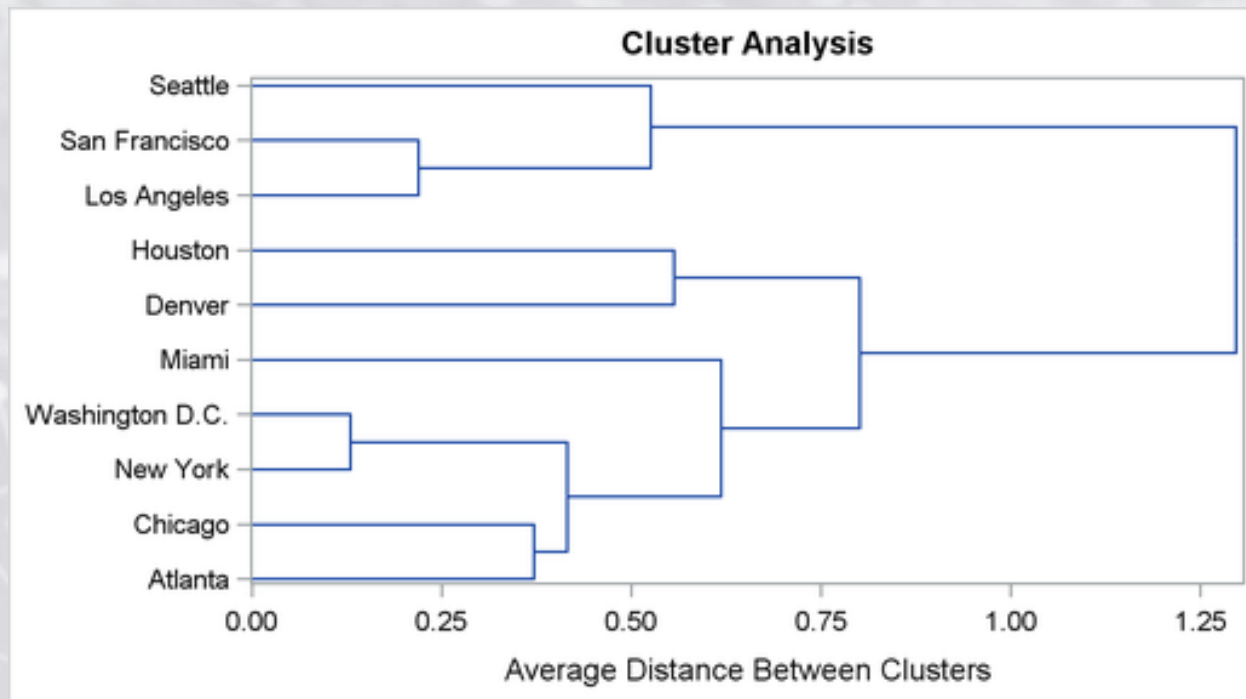


Purity as an external evaluation criterion for cluster quality. Majority class and number of members of the majority class for the three clusters are: x, 5 (cluster 1); o, 4 (cluster 2); and ◇, 3 (cluster 3). Purity is $(1/17) \times (5 + 4 + 3) \approx 0.71$.

Hierarchical Clustering

- One potential disadvantage of K-means clustering is that it requires us to pre-specify the number of clusters K .
- **Hierarchical clustering** is an alternative approach which does not require that we commit to a particular choice of K . In addition, hierarchical clustering has the advantage over K-means in that it results in an interpretable, tree-based representation of the data, called a **dendrogram**.

Hierarchical Clustering: Dendrograms



- Each *leaf* in a dendrogram represents a datum; as we traverse the tree (from left to right – or equivalently from bottom to top for vertical orientations), some leaves begin to fuse into branches.
- These fused data correspond to observations that are similar to one another. Elements that fuse early (farther to the left) are more similar than elements that fuse farther to the right

Hierarchical Clustering: Dendrograms

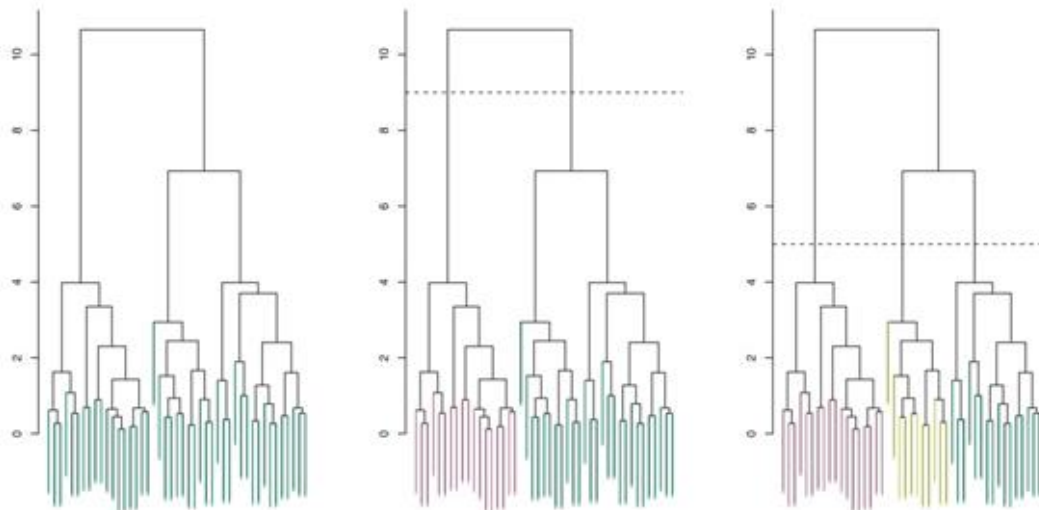
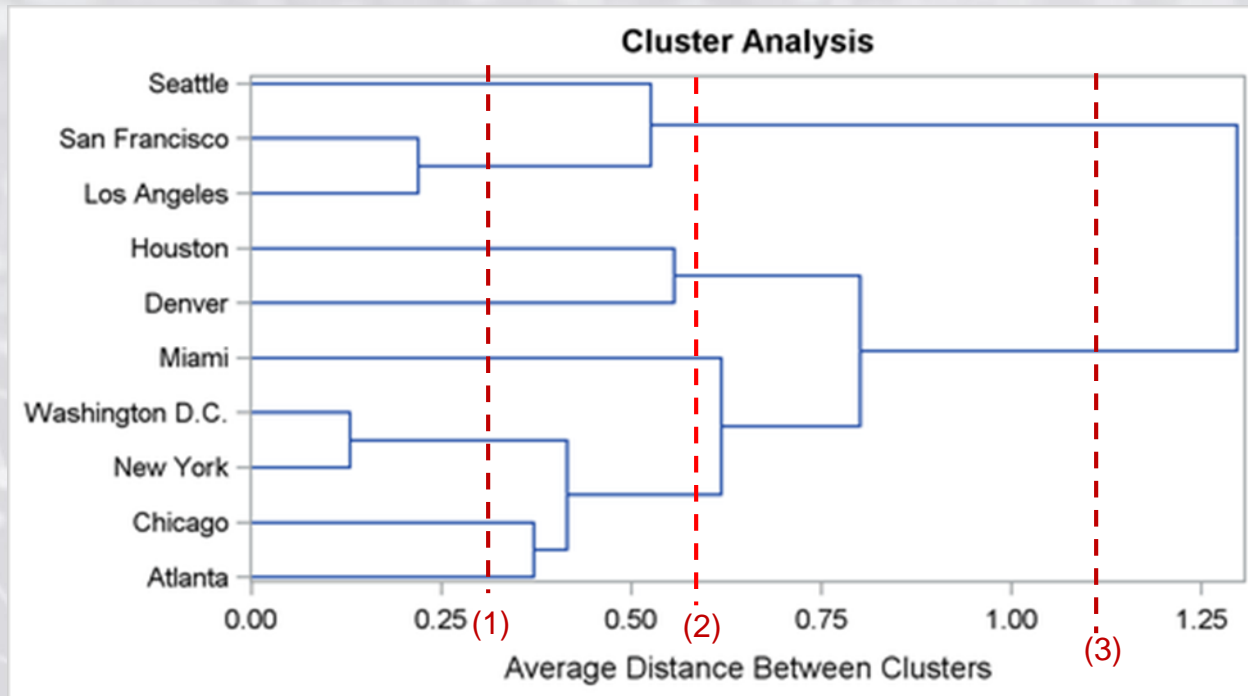


FIGURE 10.9. Left: dendrogram obtained from hierarchically clustering the data from Figure 10.8 with complete linkage and Euclidean distance. Center: the dendrogram from the left-hand panel, cut at a height of nine (indicated by the dashed line). This cut results in two distinct clusters, shown in different colors. Right: the dendrogram from the left-hand panel, now cut at a height of five. This cut results in three distinct clusters, shown in different colors. Note that the colors were not used in clustering, but are simply used for display purposes in this figure.

- One strongly attractive aspect of hierarchical clustering is that a single dendrogram can be used to obtain any number of clusters (see figure) – we slice the tree at various heights to yield different clusterings (for different numbers of clusters).
- The term *hierarchical* refers to the fact that clusters obtained by cutting the dendrogram at a given height (for a vertical orientation, as shown) are necessarily nested within the clusters obtained by cutting the dendrogram at any greater height.

Hierarchical Clustering: Dendrograms



- Example above shows three dendrogram “slices” rendering different clusterings; in **case (1)** the slice produces 8 clusters; **case (2)** produces 4 total clusters; **case (3)** generates 2 clusters.

Hierarchical Clustering Algorithm

- The hierarchical clustering dendrogram is obtained via an extremely simple (bottom-up) algorithm.
- We begin by defining some sort of *dissimilarity measure* between each pair of observations; most often a Euclidean distance is used.

(*) The algorithm proceeds iteratively, in a greedy fashion; starting at the bottom of the dendrogram, each of the n observations is treated as its own cluster. Two clusters that are most similar are fused so that now there are $n-1$ clusters. Next the two clusters that are most similar are fused again, etc. The algorithm halts when one cluster remains.

Hierarchical Clustering Algorithm

- The hierarchical clustering dendrogram is obtained via an extremely simple (bottom-up) algorithm.
- We begin by defining some sort of *dissimilarity measure* between each pair of observations; most often a Euclidean distance is used.

(*) The algorithm proceeds iteratively, in a greedy fashion; starting at the bottom of the dendrogram, each of the n observations is treated as its own cluster. Two clusters that are most similar are fused so that now there are $n-1$ clusters. Next the two clusters that are most similar are fused again, etc. The algorithm halts when one cluster remains.

Here we need to extend our concept of dissimilarity beyond pairs of observations, and to dissimilarity between clusters. This extension is achieved by developing the notion of **linkage**, which defines dissimilarity between to groups of observations.

The (4) most common types of linkage – *complete*, *average*, *single* and *centroid* are described next, along with the general hierarchical clustering algorithm.

Hierarchical Clustering Algorithm

1. Begin with n observations and a measure (such as Euclidean distance) of all the $\binom{n}{2} = n(n-1)/2$ pairwise dissimilarities. Treat each observation as its own cluster.
2. For $i = n, n-1, \dots, 2$:
 - (a) Examine all pairwise inter-cluster dissimilarities among the i clusters and identify the pair of clusters that are least dissimilar (that is, most similar). Fuse these two clusters. The dissimilarity between these two clusters indicates the height in the dendrogram at which the fusion should be placed.
 - (b) Compute the new pairwise inter-cluster dissimilarities among the $i-1$ remaining clusters.

Linkage	Description
Complete	Maximal intercluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the <i>largest</i> of these dissimilarities.
Single	Minimal intercluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the <i>smallest</i> of these dissimilarities. Single linkage can result in extended, trailing clusters in which single observations are fused one-at-a-time.
Average	Mean intercluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the <i>average</i> of these dissimilarities.
Centroid	Dissimilarity between the centroid for cluster A (a mean vector of length p) and the centroid for cluster B. Centroid linkage can result in undesirable <i>inversions</i> .

(4) Common linkage types

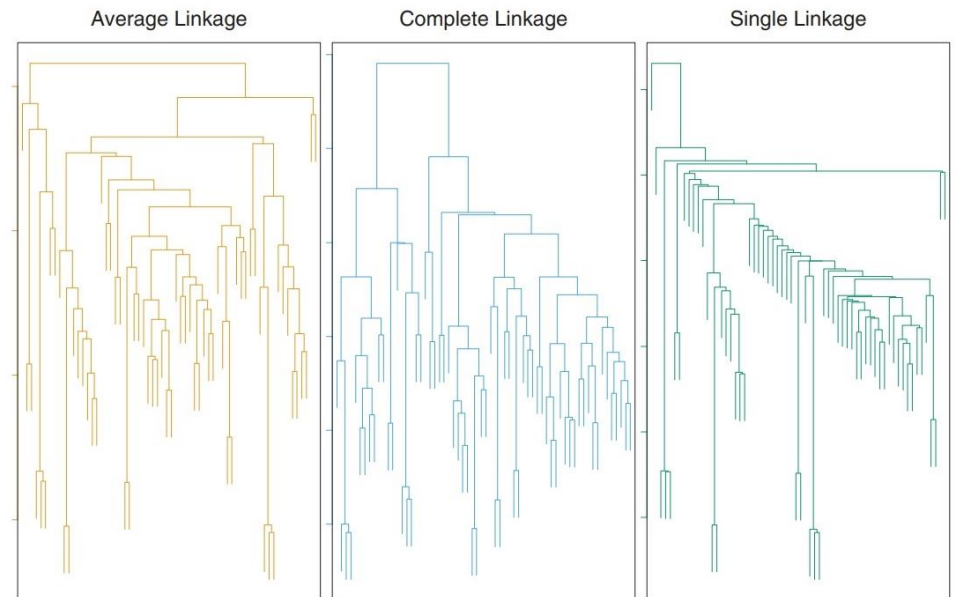
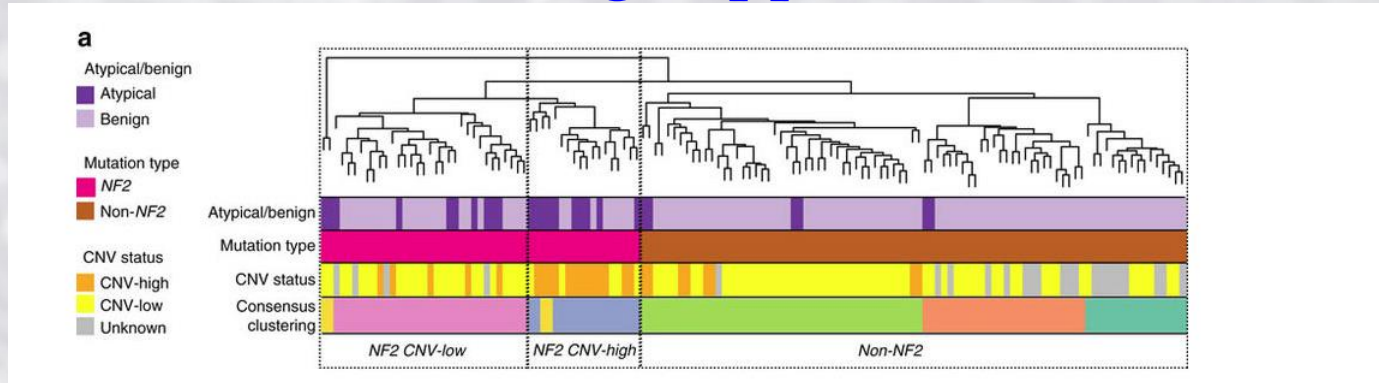


FIGURE 10.12. Average, complete, and single linkage applied to an example data set. Average and complete linkage tend to yield more balanced clusters.

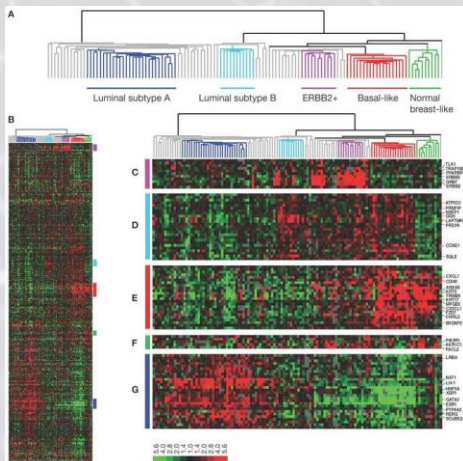
Hierarchical Clustering: Application to Genomics

- In genomics, it is frequently necessary to identify groups of genes with similar expression profiles across a large number of experiments.
- Hierarchical clustering attempts to group genes into small clusters and to group clusters into higher-level systems. The resulting hierarchical tree is easily viewed as a dendrogram.
- Most studies involve comparing a series of experiments to identify genes that are consistently coregulated under some defined set of circumstances—disease state, drug dose, etc.

Hierarchical Clustering: Application to Genomics



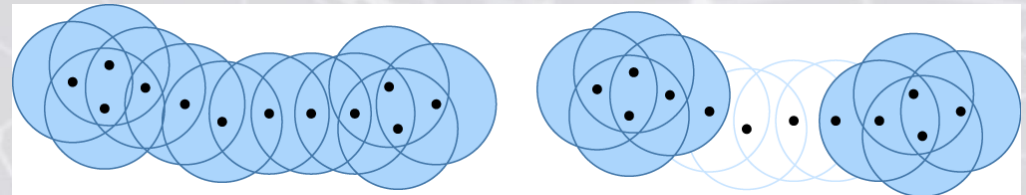
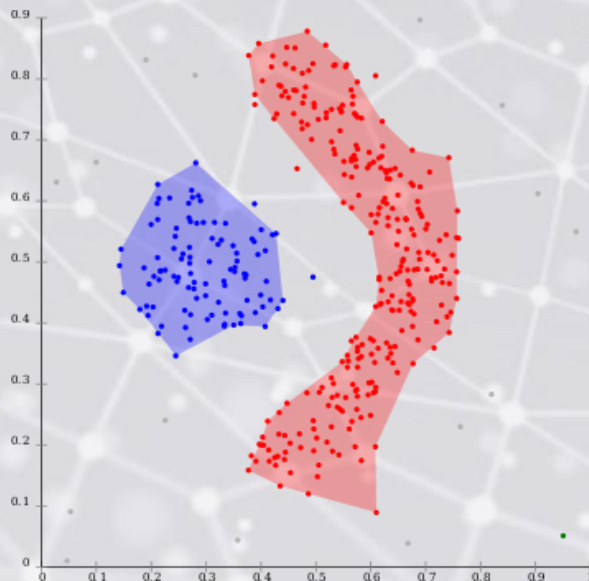
Unsupervised hierarchical clustering of 138 meningiomas by genome-wide expression profiling is shown. Atypical versus benign histology, underlying meningioma driver mutations, copy number variations, which are color coded, are shown on the left. Although the expression profile accurately clusters meningioma samples based on driver mutations, it does not fully differentiate atypical versus benign tumors. <https://www.nature.com/articles/ncomms14433/figures/2>



<https://www.nature.com/articles/ncponc0072>

DBSCAN (1996)

- **DBSCAN** (*density-based spatial clustering of applications with noise*) is one of the most popular and heavily cited clustering algorithms in scientific literature (11,000+ citations).
- It is a density-based clustering algorithm: given a set of points in some space, DBSCAN groups points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away).



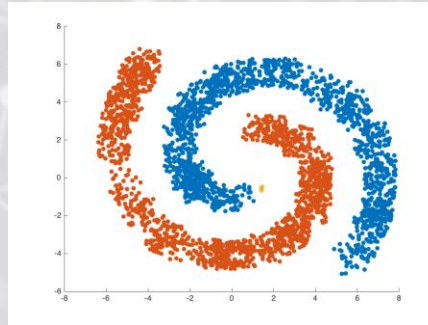
(*) Original DBSCAN paper: <https://www.aaai.org/Papers/KDD/1996/KDD96-037.pdf>

DBSCAN: Overview

- Before discussing the details of the algorithm, let's first consider some of the broader advantages/disadvantages of DBSCAN as a clustering algorithm.

Advantages:

- (*) Unlike k-means, DBSCAN does not require an *a priori* specification of the number of clusters (i.e. the hyperparameter k).
- (*) DBSCAN can find arbitrarily shaped clusters. It can even find a cluster completely surrounded by (but not connected to) a different cluster.



K-means
can't do
this!

- (*) DBSCAN is robust to outliers; and requires the specification of only two total parameters.

Disadvantages:

- (*) Algorithm is non-deterministic and often sensitive to the order the data are processed.
- (*) The quality of the clustering is highly dependent upon the distance measure/parameter values.

DBSCAN: Overview

- The basic idea behind DBSCAN is clusters are comprised of dense groupings of data points.
- The algorithm relies on two parameters: ϵ (eps) – which represents the *maximum radius* of a cluster and **MinPts** – which denotes the *minimum threshold for the number of points that form a cluster*.

The algorithm classifies all points in the data set into one of (4) categories:

DBSCAN: Overview

- The algorithm relies on two parameters: ϵ (eps) – which represents the *maximum radius* of a cluster and **MinPts** – which denotes the *minimum threshold* for the number of points that form a cluster.

The algorithm classifies all points in the data set into one of (4) categories:

(1) **core points**: a point p is a core point if at least MinPts points are within distance ϵ of it (including p itself). These points are said to be *directly reachable* from the core point p .

(2) **directly density-reachable points**: a point q is directly density-reachable from the core point p if $d(p, q) < \epsilon$.

(3) **density-reachable points**: a point q is density-reachable from the core point p if there exists a path $p \rightarrow p_2 \rightarrow p_3 \rightarrow \dots \rightarrow q$ where each p_{i+1} is directly reachable from p_i , and all points in the path are core points (with the possible exception of q).

(4) **outliers**: all points not reachable from any other points are outliers.

DBSCAN: Overview

- The algorithm relies on two parameters: ϵ (eps) – which represents the *maximum radius* of a cluster and **MinPts** – which denotes the *minimum threshold for the number of points that form a cluster*.

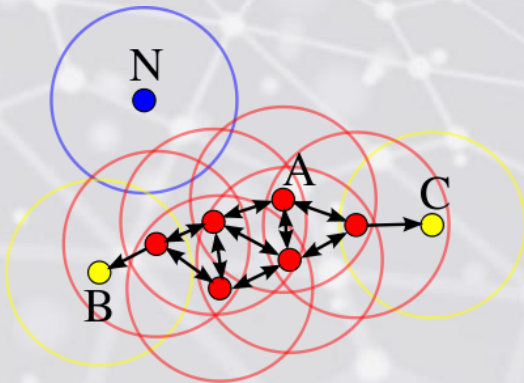
(1) **core points**: a point p is a core point if at least MinPts points are within distance ϵ of it (including p itself). These points are said to be *directly reachable* from the core point p .

(2) **directly density-reachable points**: a point q is directly density-reachable from the core point p if $d(p,q) < \epsilon$.

(3) **density-reachable points**: a point q is density-reachable from the core point p if there exists a path $p \rightarrow p_2 \rightarrow p_3 \rightarrow \dots \rightarrow q$ where each p_{i+1} is directly reachable from p_i , and all points in the path are core points (with the possible exception of q).

(4) **outliers**: all points not reachable from any other points are outliers.

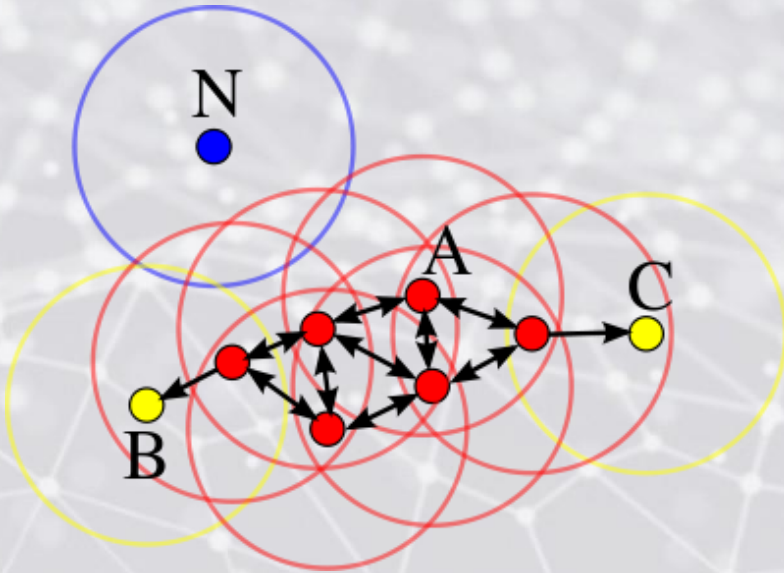
(*) **If p is a core point, then it forms a cluster together with all points (core or non-core) that are reachable from it.** Each cluster contains at least one core point; non-core points can be part of a cluster, but they form its "edge", since they cannot be used to reach more points.



In this diagram, **minPts = 4**. Point A and the other red points are core points, because the area surrounding these points in an ϵ radius contain at least 4 points (including the point itself). Because they are all reachable from one another, they form a single cluster. Points B and C are not core points, but are reachable from A (via other core points) and thus belong to the cluster as well. Point N is an outlier point that is neither a core point nor directly-reachable.

DBSCAN

(*) If p is a core point, then it forms a cluster together with all points (core or non-core) that are reachable from it. Each cluster contains at least one core point; non-core points can be part of a cluster, but they form its "edge", since they cannot be used to reach more points.



In this diagram, **minPts = 4**. Point A and the other red points are core points, because the area surrounding these points in an ϵ radius contain at least 4 points (including the point itself). Because they are all reachable from one another, they form a single cluster. Points B and C are not core points, but are reachable from A (via other core points) and thus belong to the cluster as well. Point N is an outlier point that is neither a core point nor directly-reachable.

(*) Note that **reachability is not symmetric** – since, for instance, no point may be reachable from a non-core point. However, if we defined two points p and q as **density-connected** if there is a point o such that both p and q are reachable from o , then density-connectedness is, in fact, symmetric.

A DBSCAN cluster consequently satisfies (2) properties:

- (1) All points within the cluster are mutually *density-connected*
- (2) If a point is density-reachable from any point of the cluster, it is part of the cluster as well.

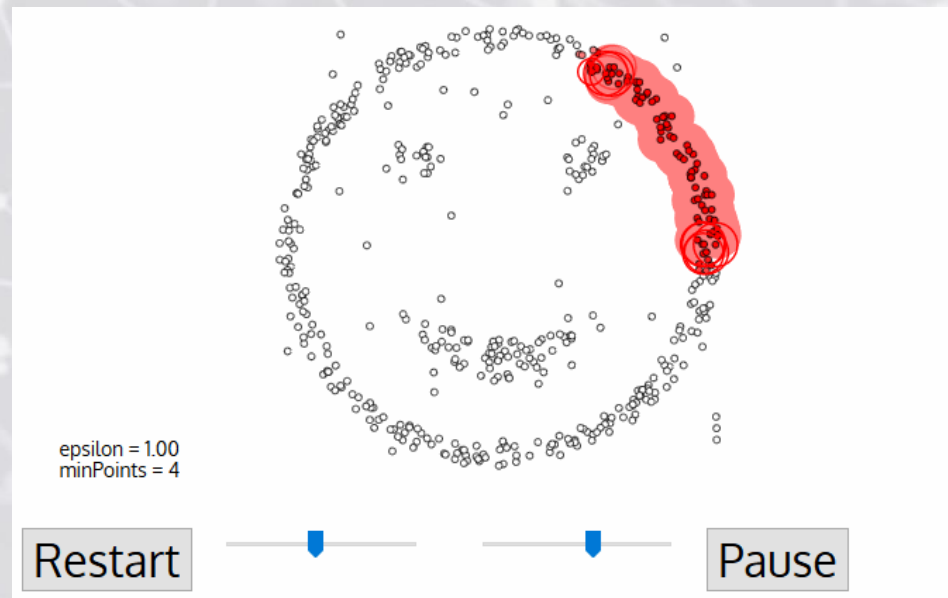
DBSCAN

High-level program execution:

- (1) Determine the parameters: $\epsilon > 0$ and MinPts (a positive natural number)
- (2) Pick an arbitrary point in the dataset; if there are more than MinPts points distance epsilon from that point (including the point itself), these points are part of a cluster.
- (3) *Expand this cluster by checking all of the new points and seeing if they too have more than minPts within distance epsilon – grow cluster recursively.

Repeat this process for new points – if a point is not part of any cluster we deem it an outlier.

*NB: The assignment of “border points” in DBSCAN is dependent upon the order in which the datapoints are processed.



DBSCAN: Pseudocode & Demo

```
DBSCAN(DB, dist, eps, minPts) {  
    C = 0  
    for each point P in database DB {  
        if label(P) ≠ undefined then continue  
        Neighbors N = RangeQuery(DB, dist, P, eps)  
        if |N| < minPts then {  
            label(P) = Noise  
            continue  
        }  
        C = C + 1  
        label(P) = C  
        Seed set S = N \ {P}  
        for each point Q in S {  
            if label(Q) = Noise then label(Q) = C  
            if label(Q) ≠ undefined then continue  
            label(Q) = C  
            Neighbors N = RangeQuery(DB, dist, Q, eps)  
            if |N| ≥ minPts then {  
                S = S ∪ N  
            }  
        }  
    }  
}
```

/ Cluster counter */*
/ Previously processed in inner loop */*
/ Find neighbors */*
/ Density check */*
/ Label as Noise */*
/ next cluster label */*
/ Label initial point */*
/ Neighbors to expand */*
/ Process every seed point */*
/ Change Noise to border point */*
/ Previously processed */*
/ Label neighbor */*
/ Find neighbors */*
/ Density check */*
/ Add new neighbors to seed set */*

DBSCAN: Analysis

Complexity:

(*) DBSCAN visits each point of the database, possibly multiple times (e.g., as candidates to different clusters).

(*) For practical considerations, however, the time complexity is mostly governed by the number of regionQuery invocations (i.e. determining the neighbors). DBSCAN executes exactly one such query for each point, and if an efficient indexing structure is used that executes a neighborhood query in $O(\log n)$, an overall average runtime complexity of $O(n \log n)$ is obtained.

(*) Without the use of an accelerating index structure the worst case run time complexity remains $O(n^2)$ (this is the result of the computational bottleneck due to brute force pairwise comparisons of points).

DBSCAN: Analysis

Parameter Tuning:

(*) Ideally, the value of ϵ is given by the problem to solve (e.g. a physical distance), and minPts is then the desired minimum cluster size.

MinPts: As a rule of thumb, a minimum minPts can be derived from the number of dimensions D in the data set, as $\text{minPts} \geq D + 1$. However, larger values are usually better for data sets with noise and will yield more significant clusters.

ϵ : if ϵ is chosen much too small, a large part of the data will not be clustered; whereas for a too high value of ϵ , clusters will merge and the majority of objects will be in the same cluster. In general, small values of ϵ are preferable, and as a rule of thumb only a small fraction of points should be within this distance of each other.

Distance function: The choice of distance function is tightly coupled to the choice of ϵ , and has a major impact on the results.

Vector Quantization

(*) It is often useful to interpret k-means as a *greedy algorithm* for approximately minimizing a loss function **related to data compression**.

- Suppose we want to perform lossy compression of some real-value vectors: $\mathbf{x}_i \in \mathbb{R}^D$.
- A simple approach to achieve this is to use **vector quantization** (VQ). The basic idea is to replace each real-valued vector \mathbf{x}_i with a discrete symbol $z_i \in \{1, \dots, K\}$, which is an index to a **codebook** of K prototypes: $\boldsymbol{\mu}_k \in \mathbb{R}^D$.
- Each data vector is encoded by using the index of the most similar prototype, where *similarity* is measured in terms of Euclidean distance:

$$\text{encode}(\mathbf{x}_i) = \arg \min_k \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$$

Vector Quantization

$$\text{encode}(\mathbf{x}_i) = \arg \min_k \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$$

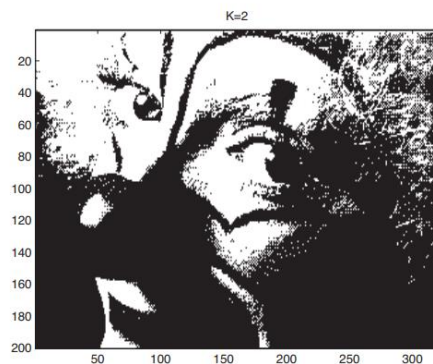
- We can define a natural cost function that measures the quality of a codebook by computing the **reconstruction error** or **distortion** it induces:

$$J(\mu, z | K, X) = \frac{1}{N} \sum_{i=1}^N \|\mathbf{x}_i - \text{decode}(\text{encode}(\mathbf{x}_i))\|^2 = \frac{1}{N} \|\mathbf{x}_i - \boldsymbol{\mu}_{z_i}\|^2$$

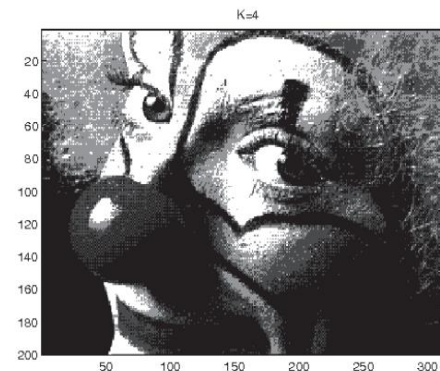
where $\text{decode}(k) = \boldsymbol{\mu}_k$. The k-means algorithm can be thought of as an iterative scheme for minimizing this objective.

Vector Quantization: Image Compression Example

- One application of VQ is image compression. Consider $N=200 \times 320=64,000$ pixel image in gray-scale so $D=1$ (shown below). If we use one byte to represent each pixel (gray-scale intensity ranges $[0,255]$), we need 512,000 bits to represent the image.
- For the compressed image we need $N \log_2 K + KC$ bits; for $K=4$ this is about 128kb, a factor of 4 compression. Greater compression could be achieved if we modelled *spatial correlation* between pixels, e.g., if we encoded 5×5 blocks (as used by jpeg). This is because the residual errors would be smaller, and would take fewer bits to encode.



(a)

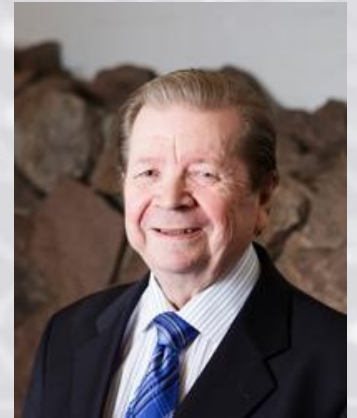


(b)

Figure 11.12 An image compressed using vector quantization with a codebook of size K . (a) $K = 2$. (b) $K = 4$. Figure generated by vqDemo.

Self-Organizing Maps (SOMs)

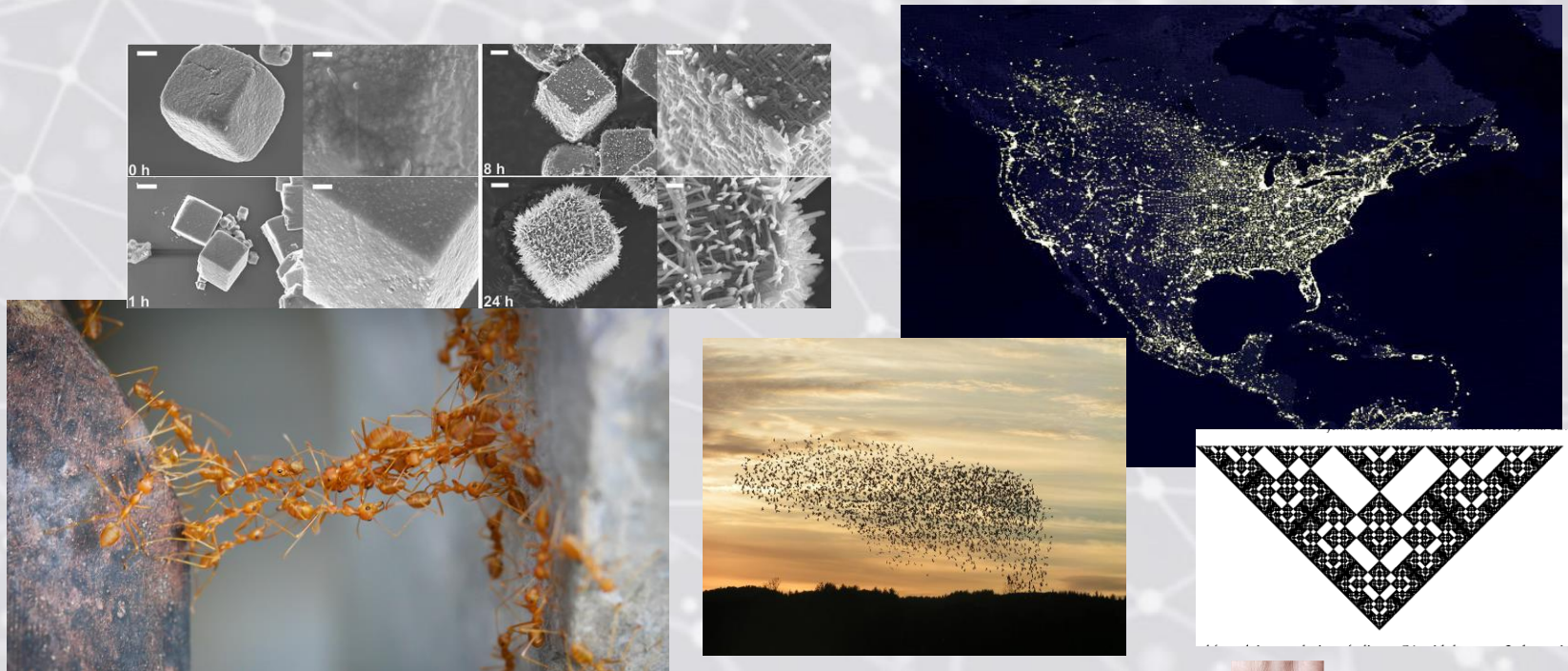
- To date, we have only considered applications of NNs for supervised learning, however, there exist several applications of NNs for unsupervised learning, including **self-organizing maps** (SOMs, 1988, Kohonen).
- In the unsupervised setting (e.g., k-means), we wish to identify meaningful data patterns in a *self-organizing fashion* (viz., Without the use of labels). This process is often referred to as learning a **feature map** – that is to say, a compression scheme that illuminates structurally significant input features.
- Stated concisely, SOMs provide a way of performing dimensionality reduction using vector quantization. Furthermore, SOMs are unique in that they preserve topographic network properties that mimic biological processes in the brain.



Self-Organization & Complex Systems

(*) **Self-organization** is a process where some form of overall order arises from local interactions between parts of an initially disordered system. The process is spontaneous, not needing control by any external agent. It is often triggered by random fluctuations, amplified by positive feedback. The resulting organization is wholly decentralized, distributed over all the components of the system. As such, the organization is typically robust and able to survive or self-repair substantial perturbation.

Self-organization occurs in many physical, chemical, biological, robotic, and cognitive systems. Systems formed from self-organization processes often exhibit **emergent behavior**.



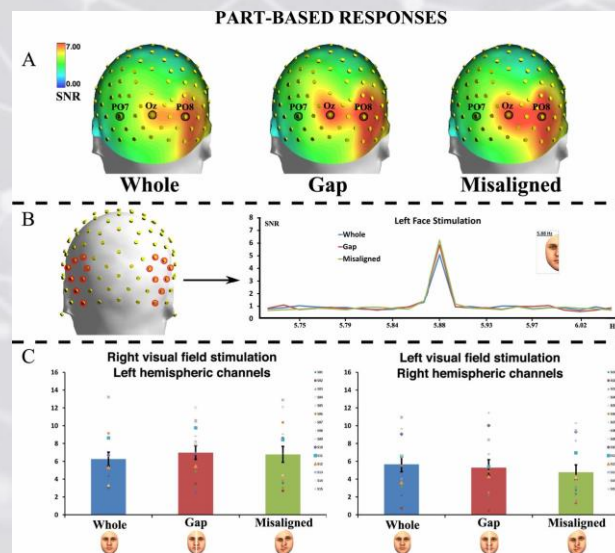
Recommended reading: M. Mitchell, *Complexity: A Guided Tour*.

Topographic Maps & The Brain

- Neurobiological studies indicate that different sensory inputs (motor, visual, auditory, etc.) are mapped onto corresponding areas of the cerebral cortex in an **orderly fashion**. This form of map, known as a **topographic map** has (2) important properties:

- (1) Each piece of information is kept in its proper context/neighborhood;
- (2) neurons dealing with closely-related pieces of information are kept close together so that they can interact using short synaptic connections.

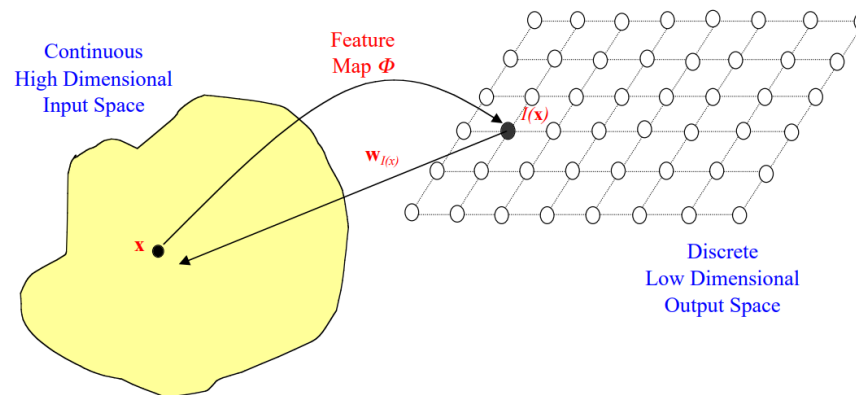
(*) SOMs train an artificial topographic map through self-organization in a neurobiologically inspired way, abiding by the **principle of topographic map formation**: “The spatial location of an output neuron in a topographic map corresponds to a particular domain or feature drawn from the input space.”



SOMs: Overview

- The goal of a SOM is to transform the incoming signal pattern into a lower dimensional discrete map, and to perform this transformation adaptively in a topographically-ordered fashion (so that neurons that are close together represent inputs that are close together, while neurons that are far apart represent inputs that are far apart).
- SOMs utilize a class of unsupervised learning techniques known as **competitive learning**, in which output neurons compete amongst themselves to be activated, with the result being that only one is activated for a given input.
- This activated neuron is called a **winner-takes-all neuron** (also: **winning neuron**). Neurons become selectively tuned to various input patterns during the course of competitive learning.

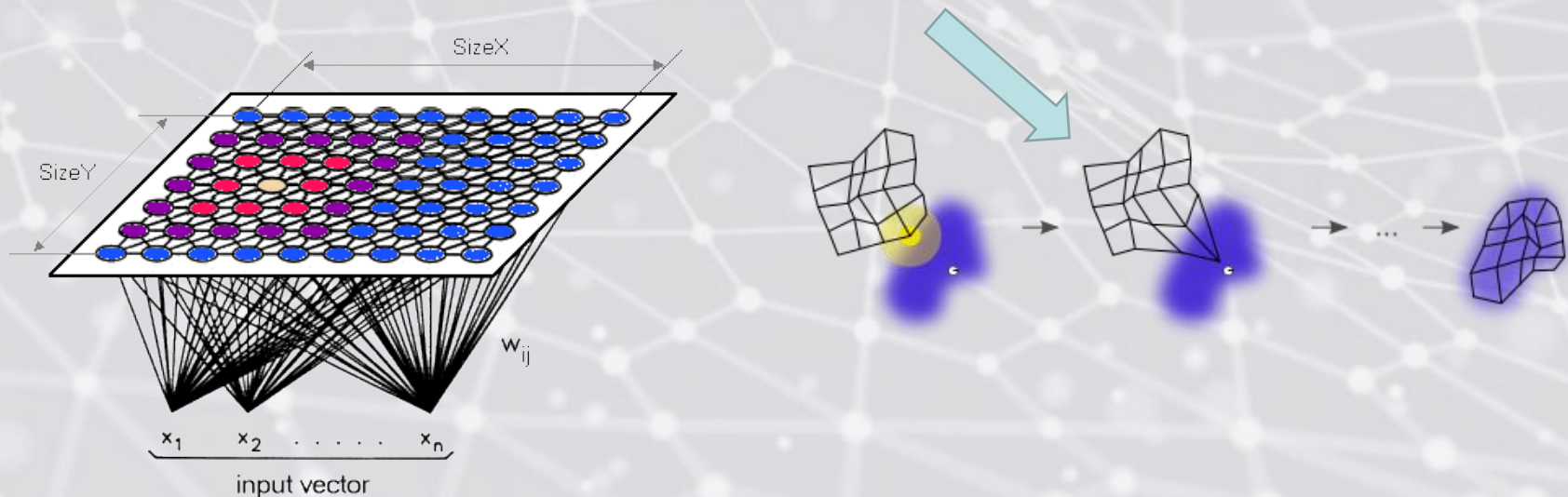
We have points \mathbf{x} in the input space mapping to points $I(\mathbf{x})$ in the output space:



Each point I in the output space will map to a corresponding point $\mathbf{w}(I)$ in the input space.

SOMs

- Note that with SOMs, the relative locations of the neurons in the network matters (nearby neurons correspond to similar input patterns) and the **neurons are arranged in a lattice/grid** (usually in 1-D or 2-D) with connections between the neurons, rather than in layers with connections only between different layers (as with the previous NNs we've seen). Each neuron is fully connected to all the source nodes in the input layer.
- Each node has a specific topological position (an (x,y) coordinate in the lattice) and contains a vector of weights.
- For training, neurons are tuned to conform with the topographic map criteria; in this way, the winning neuron should pull other neurons that are close to it in the network closer to itself in weight space, whereas neurons that are very far away should be ignored.



SOM Algorithm

The Self-Organising Feature Map Algorithm

• Initialisation

- choose a size (number of neurons) and number of dimensions d for the map
- Either:
 - * choose random values for the weight vectors so that they are all different OR
 - * set the weight values to increase in the direction of the first d principal components of the dataset

• Learning

- repeat:
 - * for each datapoint:
 - select the best-matching neuron n_b using the minimum Euclidean distance between the weights and the input,

$$n_b = \min_j \|\mathbf{x} - \mathbf{w}_j^T\|. \quad (9.8)$$

- * update the weight vector of the best-matching node using:

$$\mathbf{w}_j^T \leftarrow \mathbf{w}_j^T + \eta(t)(\mathbf{x} - \mathbf{w}_j^T), \quad (9.9)$$

where $\eta(t)$ is the learning rate.

- * update the weight vector of all other neurons using:

$$\mathbf{w}_j^T \leftarrow \mathbf{w}_j^T + \eta_n(t)h(n_b, t)(\mathbf{x} - \mathbf{w}_j^T), \quad (9.10)$$

where $\eta_n(t)$ is the learning rate for neighbourhood nodes, and $h(n_b, t)$ is the neighbourhood function, which decides whether each neuron should be included in the neighbourhood of the winning neuron (so $h = 1$ for neighbours and $h = 0$ for non-neighbours)

- * reduce the learning rates and adjust the neighbourhood function, typically by $\eta(t+1) = \alpha\eta(t)^{k/k_{\max}}$ where $0 \leq \alpha \leq 1$ decides how fast the size decreases, k is the number of iterations the algorithm has been running for, and k_{\max} is when you want the learning to stop. The same equation is used for both learning rates (η, η_n) and the neighbourhood function $h(n_b, t)$.

- until the map stops changing or some maximum number of iterations is exceeded

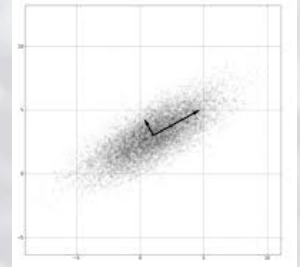
• Usage

- for each test point:
 - * select the best-matching neuron n_b using the minimum Euclidean distance between the weights and the input:

$$n_b = \min_j \|\mathbf{x} - \mathbf{w}_j^T\| \quad (9.11)$$

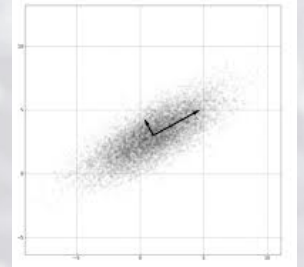
SOM Algorithm: Overview

- (I) **Initialization:** network parameters: determine number of neurons, dimension for the map (d)
-- can use a random initialization or begin with, say the PCA algorithm, using first d principal components.



SOM Algorithm: Overview

(I) **Initialization:** network parameters: determine number of neurons, dimension for the map (d)
-- can use a random initialization or begin with, say the PCA algorithm, using first d principal components.



(II) **Learning:**

(a) For each data point, *select best-matching neuron* (n_b), using minimum Euclidean distance.

(b) Update weight vector of n_b : $\mathbf{w}_j^T \leftarrow \mathbf{w}_j^T + \eta(t)(\mathbf{x} - \mathbf{w}_j^T)$

(this update has the effect of moving the weight vector of n_b closer to the datum), the learning rate $\eta(t)$ is decreased over time.



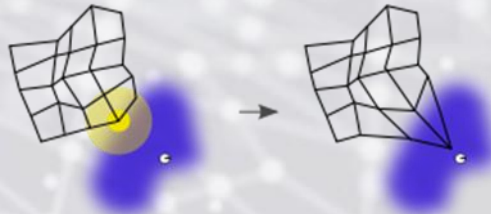
SOM Algorithm: Recap

(II) Learning:

(a) For each data point, *select best-matching neuron* (n_b), using minimum Euclidean distance.

(b) Update weight vector of n_b : $\mathbf{w}_j^T \leftarrow \mathbf{w}_j^T + \eta(t)(\mathbf{x} - \mathbf{w}_j^T)$

(this update has the effect of moving the weight vector of n_b closer to the datum), the learning rate $\eta(t)$ is decreased over time.



(c) Update the weight vector of all other neurons using: $\mathbf{w}_j^T \leftarrow \mathbf{w}_j^T + \eta_n(t)h(n_b, t)(\mathbf{x} - \mathbf{w}_j^T)$

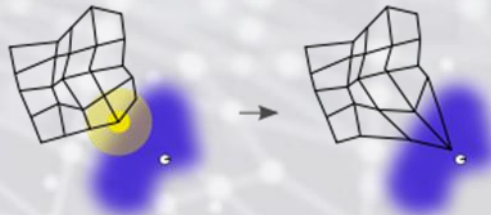
where $\eta_n(t)$ is the learning rate for the neighborhood nodes, and $h(n_b, t)$ is the **neighborhood function** with respect to node n_b , which decides whether each neuron should be included in the neighborhood of the winning neuron (e.g. $n=1$ for neighbors and $n=0$ for non-neighbors – or a Gaussian function can be used).

SOM Algorithm: Recap

(II) Learning:

(a) For each data point, *select best-matching neuron* (n_b), using minimum Euclidean distance.

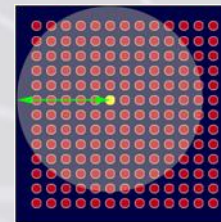
(b) Update weight vector of n_b : $\mathbf{w}_j^T \leftarrow \mathbf{w}_j^T + \eta(t)(\mathbf{x} - \mathbf{w}_j^T)$
(this update has the effect of moving the weight vector of n_b closer to the datum), the learning rate $\eta(t)$ is decreased over time.



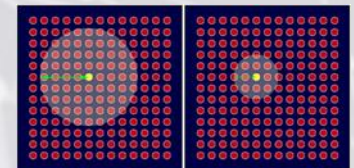
(c) Update the weight vector of all other neurons using: $\mathbf{w}_j^T \leftarrow \mathbf{w}_j^T + \eta_n(t)h(n_b, t)(\mathbf{x} - \mathbf{w}_j^T)$

where $\eta_n(t)$ is the learning rate for the neighborhood nodes, and $h(n_b, t)$ is the **neighborhood function** with respect to node n_b , which decides whether each neuron should be included in the neighborhood of the winning neuron (e.g. $n=1$ for neighbors and $n=0$ for non-neighbors – or a Gaussian function can be used).

(d) Reduce the learning rates and adjust the neighborhood function (neighborhood size decreases over time).



$h(n_b, t)$ function



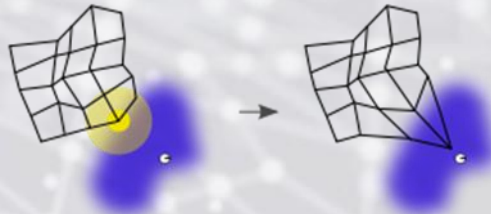
neighborhood size
decreases over time

SOM Algorithm: Recap

(II) Learning:

(a) For each data point, *select best-matching neuron* (n_b), using minimum Euclidean distance.

(b) Update weight vector of n_b : $\mathbf{w}_j^T \leftarrow \mathbf{w}_j^T + \eta(t)(\mathbf{x} - \mathbf{w}_j^T)$
(this update has the effect of moving the weight vector of n_b closer to the datum), the learning rate $\eta(t)$ is decreased over time.



(c) Update the weight vector of all other neurons using: $\mathbf{w}_j^T \leftarrow \mathbf{w}_j^T + \eta_n(t)h(n_b, t)(\mathbf{x} - \mathbf{w}_j^T)$

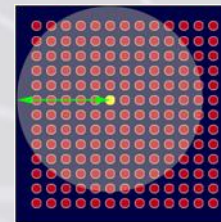
where $\eta_n(t)$ is the learning rate for the neighborhood nodes, and $h(n_b, t)$ is the **neighborhood function** with respect to node n_b , which decides whether each neuron should be included in the neighborhood of the winning neuron (e.g. $n=1$ for neighbors and $n=0$ for non-neighbors – or a Gaussian function can be used).

(d) Reduce the learning rates and adjust the neighborhood function (neighborhood size decreases over time).

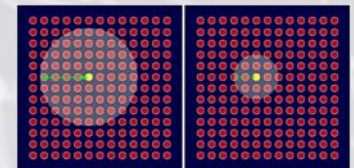
(III) Testing:

For each test point select best-matching neuron:

$$n_b = \min_j \|\mathbf{x} - \mathbf{w}_j^T\|$$

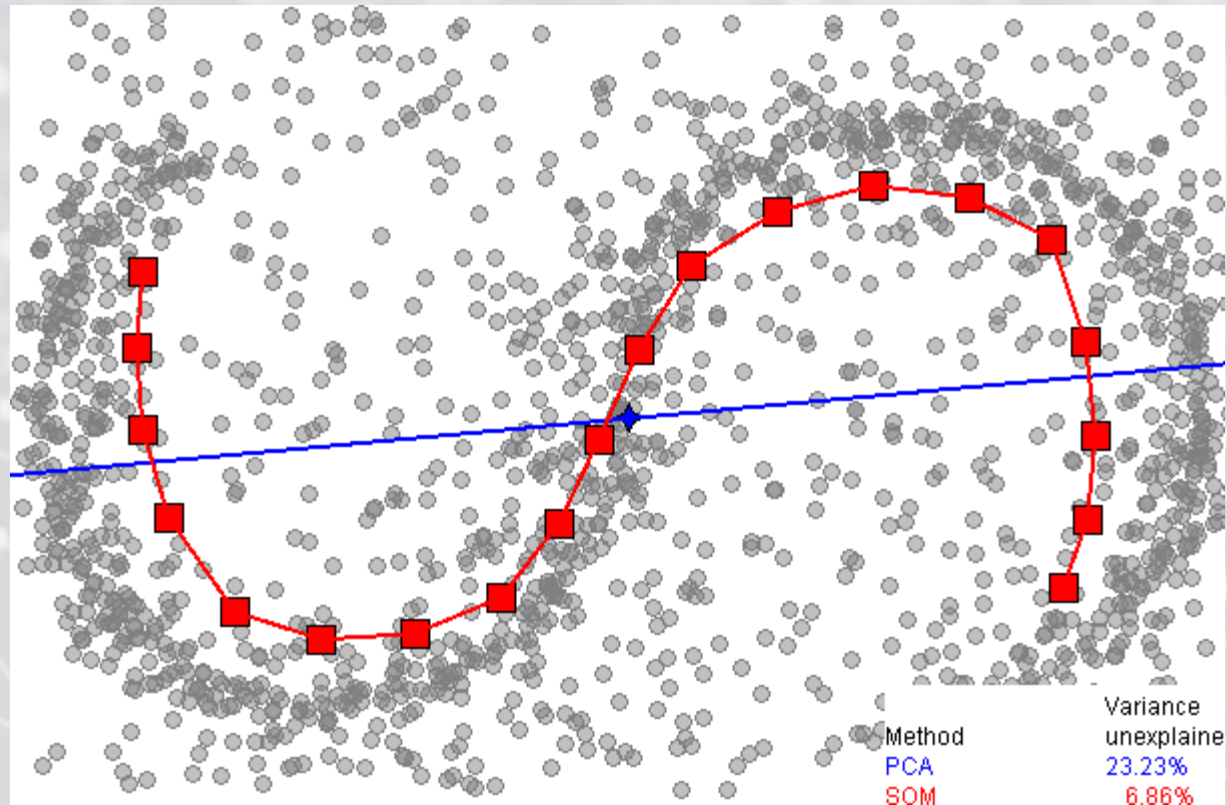


$h(n_b, t)$ function



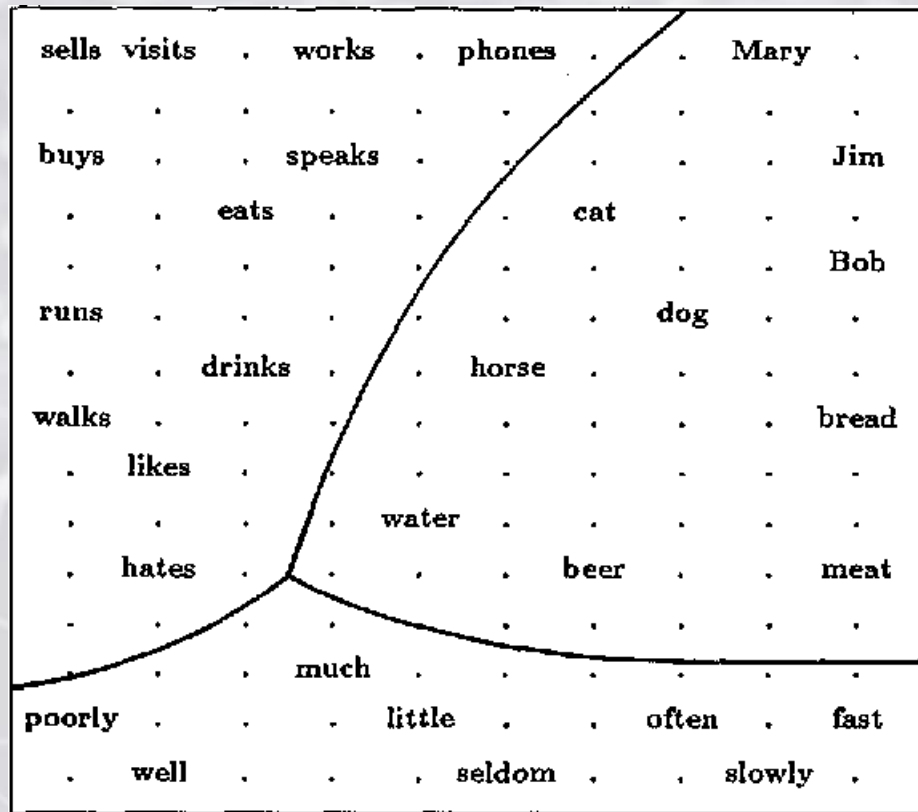
neighborhood size decreases over time

SOM vs PCA



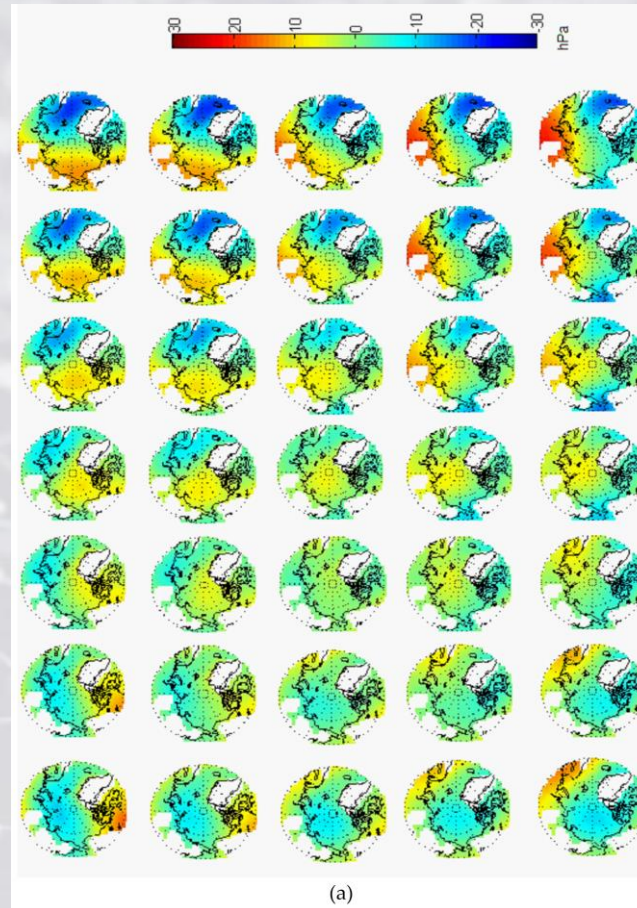
One-dimensional SOM versus principal component analysis (PCA) for data approximation. SOM is a red broken line with squares, 20 nodes. The first principal component is presented by a blue line. Data points are the small grey circles. For PCA, the fraction of variance unexplained in this example is 23.23%, for SOM it is 6.86%.

SOM for Semantic Maps



Semantic network (SOM) detects “logical similarity” between words based on statistics of their contexts (e.g. word order).

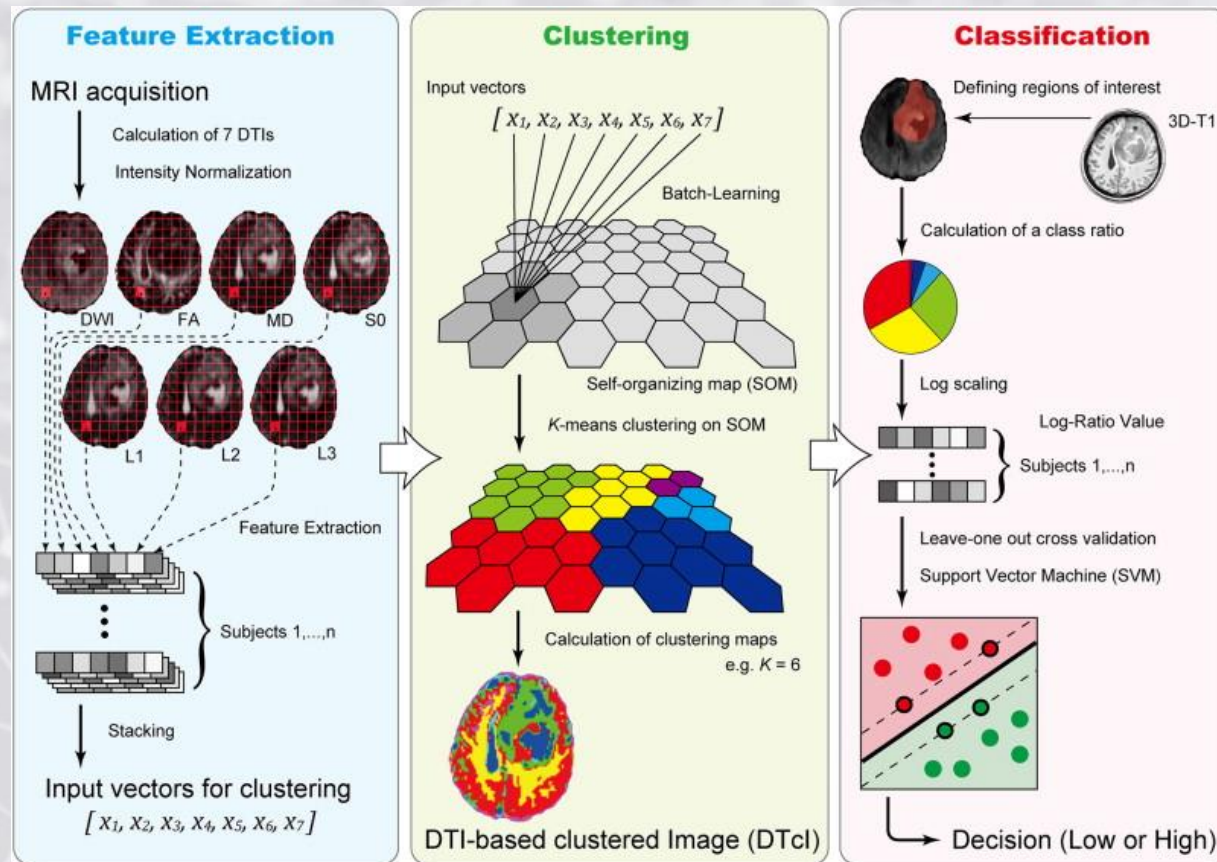
SOM for Atmospheric Science



SOM of sea level pressure anomaly patterns; different days fall into different categories, allowing researchers to attribute causes for variation with greater specificity.

<https://www.intechopen.com/books/applications-of-self-organizing-maps/self-organizing-maps-a-powerful-tool-for-the-atmospheric-sciences>

SOM for Medical Diagnosis



Pipeline used to predict glioma (tumor) grade and subsequently guide therapeutic strategies. First MRI data is acquired, the data was clustered in (2) steps beginning with an SOM, followed by k-means; lastly classification between high and low gliomas was done using an SVM.

Fin

